BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

LATTICE-BASED SEARCH STRATEGIES FOR LARGE VOCABULARY SPEECH RECOGNITION

by

FREDERICK RICHARDSON

B.S., Boston University, 1992.

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

1995

Approved by

First Reader

Dr. Mari Ostendorf, Associate Professor,
Department of Electrical, Computer and Systems Engineering
Boston University

Second Reader

Dr. J. Robin Rohlicek, Manager, Research and Development,
BBN Hark Systems Corp.
Research Associate,
Department of Electrical, Computer and Systems Engineering
Boston University

Third Reader

Rich Schwartz, Senior Scientist,
Bolt Beranek and Newman Inc.
Boston University

Fourth Reader

Dr. David Castanon, Associate Professor,
Department of Electrical, Computer and Systems Engineering
Boston University

Acknowledgments

I would like to first acknowledge my advisor, Prof. Mari Ostendorf, for her constant support and guidance through the research and writing which constitute this thesis. It is a great understatement to say that this work would not have been possible without her help. I would also like to acknowledge my readers, Dr. Robin Rohlicek, Rich Schwartz and Prof. David Castanon, for their insightful comments and helpful discussions.

I would like to thank my lab cohorts, Sanjay, Rukmini and Owen, for there help and tolerance throughout this thesis process. In particular, I would like to thank Owen for helping me understand the detailed functions of the original BU recognition system which he built from scratch.

I would also like to thank my family who has been extremely supportive throughout the time I've been working on this thesis. My brother Jon was always a phone call away and my parents were always there when I needed them.

Finally I would like to acknowledge support from ONR under contract number N00014-92-J-1778.

LATTICE-BASED SEARCH STRATEGIES
FOR LARGE VOCABULARY SPEECH RECOGNITION

(Order No.)

FREDERICK S. RICHARDSON

Boston University, College of Engineering, 1995

Major Professor: Mari Ostendorf,
Associate Professor of Electrical, Computer
and Systems Engineering

Abstract

The design of search algorithms is an important issue in recognition, particularly for very large vocabulary, continuous speech. It is an especially crucial problem when computationally expensive knowledge sources are used in the system, as is necessary to achieve high accuracy. Recently, multi-pass search strategies have been used as a means of applying inexpensive knowledge sources early on to prune the search space for subsequent passes using more expensive knowledge sources. Three multi-pass search algorithms are investigated in this thesis work: the N-best search algorithm, a lattice dynamic programming search algorithm and a lattice local search algorithm. Both the lattice dynamic programming and lattice local search algorithms are shown to achieve comparable performance to the N-best search algorithm while running as much as 10 times faster on a 20,000 word vocabulary task. The lattice local search algorithm is also shown to have the additional advantage over the lattice dynamic programming search algorithm of allowing sentence-level knowledge sources to be incorporated into the search.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of much development of speech recognition technology is to allow people to interact with machines using speech. This is an important problem because there are many circumstances in which a person needs to interact with a machine but cannot use their hands to control input devices. For example, a disabled person may need to use a computer but may not be able to use a mouse or keyboard, or a repair technician may need to access a computer during a difficult procedure which requires the use of both hands. There are also many other examples of applications of speech recognition technology to education, language translation and computer information access over the phone. Clearly, the speech recognition problem is a very important one and, if solved, could improve many aspects of our lives.

Although speech recognition technology has started to appear in high end and even affordable personal computer systems, the technology has not yet reached a point where a person can interact with a computer by only using their voice. This is despite the fact that many important advances have been made over the past few decades of speech recognition research. During this time, the speech recognition problem has proven to be a difficult one, and, in order to tackle it more effectively, it has been partitioned into several different areas of focus including speech recognition search, acoustic modeling and language modeling.

One important problem in speech recognition is how to find the best sentence hypothesis for an utterance given various information sources for evaluating different hypotheses. This is the speech recognition search problem. To solve this problem, all sentence hypotheses must be evaluated over all time alignments with the acoustic signal. The search problem is a difficult one because, in general, it is impractical to compare all possible hypotheses for an utterance due to the amount of time and storage that is required. Therefore, the goal for speech recognition search is to design an algorithm that is both efficient (in terms of storage and time) and accurate (in terms of minimizing search errors). A search error occurs when a sentence hypothesis is not considered by the search algorithm but it would actually score higher than the best hypothesis found for the given knowledge sources.

State-of-the-art speech recognition systems today are able to perform speaker independent recognition for vocabularies of 20,000 or more words. For a typical speech recognition system, the search computation grows on the order of V^n where V is the size of the vocabulary and n depends on the word sequence dependence represented in the language model. For example for a typical speech recognition system using a 20,000 word vocabulary and a trigram language model, the full search computation is potentially on the order of $(20,000)^3 = 8 \times 10^{12}$ times the length of the utterance, or roughly $\mathrm{O}(V^2)$ for a trigram LM in practice. The computation is reduced by using pruning techniques during the search, but the size of the vocabulary still has a profound effect on the speed of a search algorithm.

Search techniques for speech recognition have long been an important research area, because the search problem is fundamentally linked to the type of modeling assumptions made. The design of a search algorithm determines how various models and techniques for comparing different word or sentence hypotheses will be applied to find a “best” sentence hypothesis given a speech utterance.

Speech recognition search is still an important problem today. Over the past few years, the vocabulary size for speech recognition has grown from 1,000 words to 50,000 words or more. This has increased the search computation mentioned above by a factor of at least $50^2 = 2500$, assuming no pruning is used for the search and excluding the additional costs due to advances in modeling. With this increase in computational requirements, new approaches have to be considered for designing a speech recognition search algorithm. This is despite advances in the computational capacity of computers which have been increasing by about a factor of two every two years.

One approach that has been used for incorporating sentence-level or expensive models into the recognition search has been the N-best search paradigm. In this paradigm, a list of sentences is generated for an utterance by one system and the list is re-evaluated by another system. Then the overall best sentence out of the list is selected as the output of the system. The problem with this approach is that the N-best list needs to be long if the vocabulary is large or if the error rates are high, in order to increase the chances that the second system will have the opportunity to pick the highest scoring hypothesis. In any multi-pass speech recognition system, the first pass should ideally keep the answer that would be considered the “best” hypothesis by later passes, otherwise a search error has been made. Unfortunately, evaluating the hypotheses in the list becomes more expensive for a longer list. Therefore, the approach taken in this thesis work is to use a different representation of sentence hypotheses with the goal of solving two problems: 1) increasing the chance that the highest scoring sentence is available to the second pass scoring algorithm and 2) decreasing the computation of the second pass algorithm.

In this work, a word lattice is a directed graph of words with a unique start and end node and implicit time information. A sentence hypothesis is a path from the start node to the end node through the lattice. Word lattices are an efficient means of representing the same information that is in an N-best list, in terms of storage. This is especially true if the hypotheses in the N-best list only vary by one or two words. Also, it will be shown in this thesis that there are algorithms that can be used for searching lattices which are much more efficient than the comparable algorithm used in the N-best search paradigm.

Two lattice search algorithms are investigated in this work. The first algorithm finds the optimal answer in the lattice using **dynamic programming** (DP), a breadth first search on the entire lattice. The second algorithm performs a suboptimal search on the lattices using an iterative **local search** algorithm, that evaluates entire sentence hypotheses in the lattice. The key advantage of the lattice local search over the lattice DP algorithm is that it allows sentence-level knowledge sources to be used in the search. Only Markov knowledge sources can be used with the lattice DP algorithm.

There are several key contributions presented in this thesis work. The BBN decoder was modified to produce lattices with acoustic scores and phonetic segmentations. A lattice file format specification, included in the appendix of this thesis, was defined that is being considered as a standard by the speech recognition community. Two lattice search algorithms were shown to be a factor of ten faster than N-best rescoring and one of them, the lattice local search algorithm, allows for the incorporation of sentence-level knowledge sources. Also, the Boston University mixture language model, which is a sentence-level knowledge source, was incorporated into the lattice local search.

The thesis is organized as follows: Chapter 2 provides background information on the search problem in speech recognition, and discusses word and phone representations, search strategies, the N-best paradigm and local search. Then Chapter 3 outlines the speech corpora used for evaluating the different lattice-based search algorithms and the modifications that were made to the BBN decoder in order to produce lattices that were used for this work. The formulation for the lattice DP search algorithm, and the approach taken for N-best rescoring with the lattices are presented in Chapter 4. The lattice local search algorithm is then described in Chapter 5, along with the definition of the local neighborhood used for the search and algorithms for local path scoring and score caching. Chapter 6 gives experimental results for the different algorithms, and finally Chapter 7 gives an overview of the significance of this work and possible future directions.

Chapter 2

Background

This chapter gives an overview of a recognition system and the different types of knowledge sources and search strategies that are commonly used today. First, we overview a generic speech recognition system. The concept of a knowledge source is defined, and the criterion for finding the best sentence given a set of knowledge sources is given. Then the two main types of knowledge sources, language and acoustic models, are discussed followed by a description of the principle search constraints. Two types of one-pass search algorithms, the breadth-first beam search and the depth-first stack search are also presented. In the second half, we describe several algorithms that are used in the multi-pass search paradigms. The N-best rescoring paradigm is outlined along with various N-best search algorithms. Following that, the term "lattice" and "N-best lattice" are defined and lattice-based search algorithms are discussed. Finally, some details about local search algorithms are presented.

2.1 Overview of a Speech Recognition System

Speech recognition involves a search process that incorporates information from a

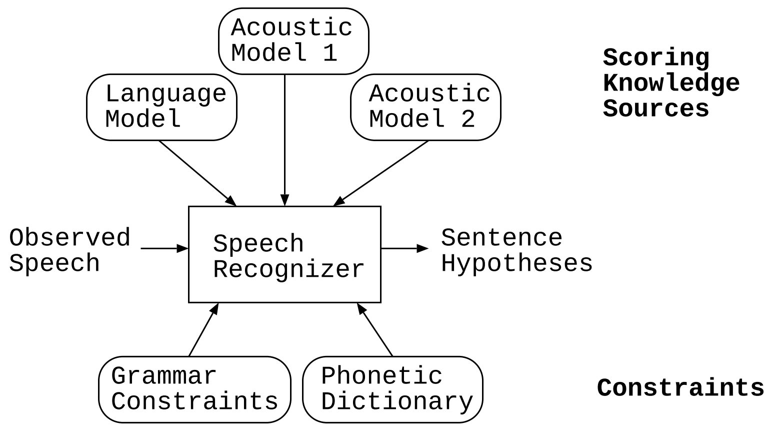


Figure 2.1: Block diagram of a speech recognizer and the information sources that it may rely on.

variety of different sources. There are two types of information sources that are used: those which provide constraints for the search, such as phonetic dictionaries and word grammars, and those which provide scores for different word sequence hypotheses, such as language and acoustic models (Figure 2.1). The second type of information source, which provides scores, will be referred to as a "knowledge source" throughout the thesis. The constraints can be thought of as knowledge sources which assign probability zero to specific sequences. However, they are often used in a deterministic structural manner which is why we consider them separately.

2.1.1 Knowledge Sources

For statistical approaches to speech recognition, the goal is to find

$$\underset{\overline{W}}{\operatorname{argmax}} \, \mathrm{P}(\overline{W}|\overline{X}) = \underset{\overline{W}}{\operatorname{argmax}} \{\log P(\overline{X}|\overline{W}) + \log P(\overline{W})\} \tag{2.1}$$

where $\overline{W}$ is a sequence of words and $\overline{X}$ is the acoustic data for the utterance. The log probabilities can be thought of as scores. The score $\log P(\overline{X}|\overline{W})$ is found using an acoustic model, while the score $\log P(\overline{W})$, is found using a language model. Often, other knowledge sources are also included in Equation 2.1 and weights are used for linearly combining them. A more general equation for finding the maximum combined score is then given by:

$$\max_{\overline{W}}\{\sum_{i=1}^{K}\lambda_i\mathbf{KS}_i(\overline{W},\overline{X})\} \tag{2.2}$$

where $\{\mathbf{KS}_i\}$ are knowledge sources which produce a score (which is generally a log probability) for the word sequence and acoustic observations and $\{\lambda_i\}$ are the weights for combining these scores.

Some knowledge sources used for speech recognition are Markov, that is they produce a score for a word or sub-word unit dependent on a fixed-length history of words or sub-word units. Other types of knowledge sources require information about the whole utterance such as sentence- or phrase-level models. The type of search algorithm that can be used for recognition is dependent on the type of knowledge sources that are being combined. For example, if only Markov knowledge sources are combined, then an optimal shortest path algorithm such as dynamic programming can be used for the search. However, if sentence- or phrase-level knowledge sources are used then an optimal search must consider each possible hypothesis separately, which is impractical.

Language Model

An important category of language models (LM's) is statistical language models. One type of statistical model, the n-gram LM, models the probability of a word given the previous $n-1$ words. Then the Markov property is used to obtain the probability of the whole utterance. The most popular types of n-gram LM's are the trigram LM ($n = 3$) and the bigram LM ($n = 2$). Since n-gram LM's are Markov, they can be used with an efficient optimal search algorithm so long as the other knowledge sources used for recognition are also Markov.

Other types of statistical, but non-Markov language models include long distance LM's which require knowledge of the entire sequence of words corresponding to an utterance. Some typical examples of long distance LM's include trigger pair LM's [1] and the Boston University sentence-level mixture model [2]. Since these LM's are not Markov, they cannot be used with an efficient optimal search algorithm.

Acoustic Model

The acoustic model gives a score dependent on the observed speech data and a hypothesized word or sub-word unit. Word acoustic models give a score for a whole word for some start and end time, while phonetic (or sub-word) acoustic models give a score for a phoneme for some start and end time. Phonetic acoustic models have the advantage over word models of being more robust when the recognition vocabulary is large. This is because each phone will have many observations in the training data since they are shared by different words.

Phonetic acoustic models can be broken down into two categories: context-dependent and context-independent models. Context-dependent models represent the probability of the acoustic observations given a phone and one or more of its neighbors, so there may be several different models for each phone (there are approximately fifty different phones used in this work). Context-independent models represent the probability of the observations given a phone assuming the phones are independent, in which case there is one model per phone which is independent of the phone's neighbors. The most common context-dependent models depend on a phone's right and left context and are referred to as "triphone models". Two examples of phonetic acoustic models are the hidden Markov model (HMM) [3] and the stochastic segment model (SSM) [4, 5].

An HMM consists of a set of states, where each state i corresponds to a specific

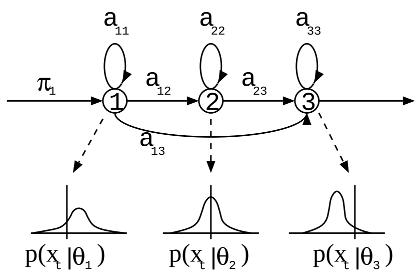


Figure 2.2: Example of an HMM state topology for a three state HMM.

probability distribution $p(y_t|\Theta_i^{\alpha_l[\alpha]\alpha_r})$ for a frame of speech y_t and set of model parameters $\Theta_i^{\alpha_l[\alpha]\alpha_r}$. An HMM has a finite-state grammar which determines the possible sequence of states that can be aligned with the speech frames within a phoneme's segment. In addition, an HMM has state transition probabilities $a_{ij} = p(i|j)$ which are associated with the HMM's state topology. The initial transition probabilities into the HMM are $\pi_i = p(i)$. Figure 2.2 is an example of a three state HMM annotated with the state transition probabilities. Note that the pairs of states which are not connected by an arc have a transition probability $a_{ij} = 0$. For the general HMM formulation, it is assumed that the frames of speech are conditionally independent given the state sequence.

A phone segment is scored using an HMM by finding the most likely state alignment for the speech frames within the segment and the corresponding segment likelihood. Given the segment of speech frames $Y_1^T = \{y_1, ..., y_T\}$ of length T, the segment score using an HMM is found with the following dynamic programming algorithm [6]:

1. Initialize: $J^*(0, j) = \pi_j$

2. For $t = 1, \ldots, T$

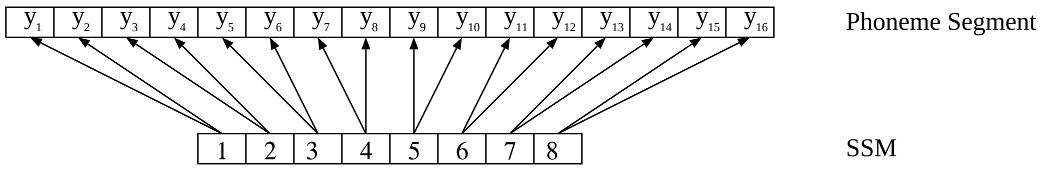


Figure 2.3: Example of an independent-frame SSM with eight distributions and the assignment of distributions to a sixteen frame phoneme segment using a linear warping.

$$J^*(t,j) \quad = \quad \max_i \quad \{J^*(t-1,i)a_{ij}p(y_t|\Theta_j^{\alpha_l[\alpha]\alpha_r})\} \qquad (2.3)$$

where $J^*(t,j)$ is the maximum probability that Y_1^t is associated with a state sequence ending with state j.

Note that with an HMM, since DP is used to find the most likely state sequence, the phone segment boundaries do not need to be hypothesized ahead of time. In fact, at each iteration of Equation 2.3, information is only required about the previous time frame and all possible previous states. The beginning time for the segment is not used explicitly. A traceback through the sequence of most likely states must be performed in order to find the best start time for a segment given a hypothesized end time.

The independent-frame SSM used for this thesis consists of eight distributions that are assigned to frames of speech within a phoneme's segment using a linear warping [4]. The model assumes that the frames of speech are conditionally independent given the warping and the segment length. Unlike the HMM, both the begin and end times of a phoneme segment must be hypothesized in order to score a segment with the SSM. Figure 2.3 is an example of the eight distribution SSM and the assignment of its distributions to a sixteen frame phoneme segment. Given a segment of speech frames $Y_1^T = \{y_1, ..., y_T\}$, the total likelihood of the segment using the SSM is found by:

$$\mathrm{P}(\mathbf{Y}_1^T|\Theta^{\alpha_l[\alpha]\alpha_r}) = \prod_{t=1}^{T} p(y_t|\Theta_{\Lambda_T(t)}^{\alpha_l[\alpha]\alpha_r})$$

where $\Theta^{\alpha_l[\alpha]\alpha_r}$ is the SSM for the triphone α with left context α_l and right context α_r. $\Theta_j^{\alpha_l[\alpha]\alpha_r}$ indicates the jth SSM model parameters, and $\Lambda_T(i)$ is the warping function which returns an SSM distribution index for each time i within a segment of length T. For this thesis, the SSM triphone acoustic score, or the log likelihood given the triphone SSM model, is used as a knowledge source and is defined as

$$\mathcal{L}(\alpha_l[\alpha]\alpha_r, t, \tau) = ln[\mathrm{P}(\mathbf{Y}_t^\tau|\theta^{\alpha_r[\alpha]\alpha_l})].$$

2.1.2 Search Constraints

Grammar

Grammars provide a means of constraining the word sequences that a recognition system can hypothesize. A finite-state grammar is a directed graph of possible word transitions that spans the entire utterance and specifies allowable word transitions. The arcs of a finite-state grammar can be assigned probabilities to specify the topology of an LM. Also, a finite-state grammar can be constructed from an n-gram LM. For example, for a bigram LM and a vocabulary of size V, the finite-state grammar consists of V nodes and V^2 arcs connecting each node to every other node.

A word-pair grammar dictates the possible words that can follow each word in the lexicon and is a special case of a finite-state grammar. A word-pair grammar is similar to the finite-state grammar corresponding to a bigram LM, except that some word transitions may not be allowed. Therefore, there may be fewer than V^2 arcs in a word-pair grammar for a vocabulary of size V.

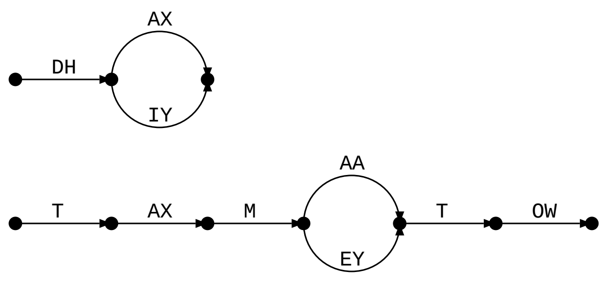


Figure 2.4: Phonetic pronunciation networks (PPN's) for the words "the" and "tomato".

Dictionary

In order to use phonetic acoustic models in speech recognition, a word dictionary which provides a mapping from a lexicon of words to a set of phonetic pronunciations is needed. One way of representing a word's pronunciation is to use an acyclic network with a unique start and end node where all path's through the network represent the set of possible pronunciations. The network must be acyclic in order to prohibit a phone from occuring an indefinite number of times in a pronunciation. A phonetic pronunciation network (PPN) is defined by the set $\{n_s^p, n_t^p, N^p, A\}$ where n_s^p and n_t^p are the PPN's source and terminal nodes, N^p is a set of all the nodes $\{n_0, \ldots, n_{|N^p|-1}\}$ and A is the set of all phone arcs $\{\alpha(n_i, n_j) | n_i, n_j \in N^p\}$. Figure 2.4 depicts the PPN's for the words "the" and "tomato".

Context-dependent models capture more detail than context-independent models and generally achieve better performance. In order to use context-dependent models for recognition, a word's PPN must be expanded to include the contexts of the right or left phones. Each arc of the triphone-expanded PPN then has a right and left context $\alpha_l[\alpha]\alpha_r(n_i, n_j)$. For example, Figure 2.5 has the same PPN's that are in Figure 2.4 expanded for triphone contexts.

Note that the triphones coming out of the source node of each PPN have no left

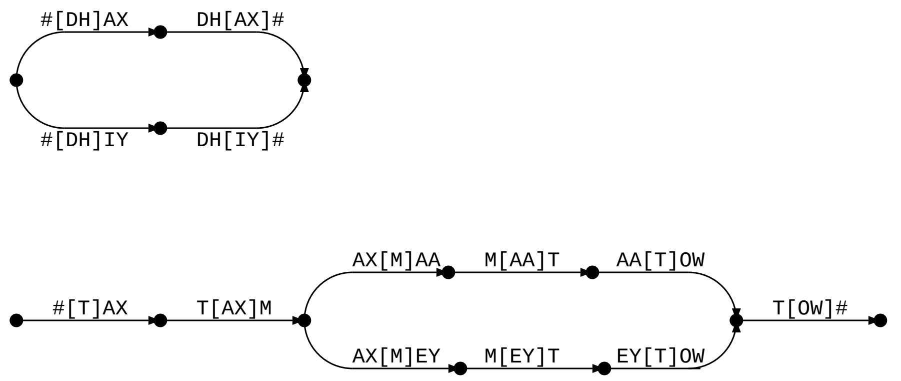


Figure 2.5: Triphone expansion of the PPN's of Figure 2.2.

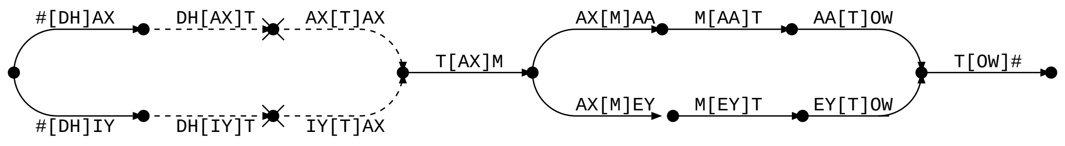


Figure 2.6: Triphone network for the word sequence "the tomato".

context and the triphones going into the terminal node of each PPN have no right context (the phone "#" is used to indicate a word beginning or ending). If triphones are going to be used to model the phonetic contexts between words, then the arcs coming out of the source node and going in to the terminal node must be duplicated for each cross-word context. Figure 2.6 depicts an example of cross word triphone arcs between the words "the" and "tomato" where the $\times$ indicates the word transition nodes. The cross-word triphone arcs are represented with dashed lines.

The search space for speech recognition is built by expanding each word in a finite-state grammar with its PPN. The type of finite-state grammar that is expanded depends on the size of n for the n-gram LM used in the search. Also, each PPN must be expanded if context-dependent acoustic models are being used. Therefore, the size of the search space is dependent on the size of the vocabulary, the order of the n-gram model and the type of acoustic model that are used for recognition.

2.1.3 Scoring and 1-Pass Search

Word Scoring

For this thesis work, the most likely segmentation and pronunciation of a word is sought. Therefore, a maximization is taken over all segmentations and pronunciations of a word. If the total probability of a word was required, then the sum would be taken over all pronunciations and segmentations instead of the maximum.

Given an observation sequence $\{y_0, \ldots, y_{T-1}\}$ corresponding to a whole word, finding the most likely pronunciation $\underline{\alpha_i} = \{\alpha_{i,1}, \ldots, \alpha_{i,l_i}\}$ and segmentation $S_0^{T-1} = \{(\tau_0 = 0, \tau_1), (\tau_1 + 1, \tau_2), \ldots, (\tau_{l_i - 1} + 1, \tau_{l_i} = T - 1)\}$ for a word w with a topologically sorted PPN $\{n_s^p, n_t^p, N^p, A\}$ (so that no node is reached before its predecessors have been scored) requires solving

$$\mathcal{L}_w(w, 0, T-1) = \max_{\underline{\alpha_i}, S_0^{T-1}} \sum_{j=1}^{l_i} \mathcal{L}(\alpha, \tau_{j-1} + 1, \tau_j)$$

which can be accomplished jointly by using the following dynamic programming (DP) iteration:

1. Initialize: $J^*(0, n_s^p) = 0$

2. For $t = 1, \ldots, T-1$

   For each $n_j \in \{N^p - n_s^p\}$ calculate:

$$\begin{aligned} J^*(t, n_j) \quad = \quad & \max_{\tau \in R(t,\alpha), \alpha_l[\alpha]\alpha_r(n_i,n_j) \in A} \quad \{J^*(\tau, n_i) + \lambda_a \mathcal{L}(\alpha_l[\alpha]\alpha_r(n_i, n_j), \tau, t) \\ & + \lambda_d \log \mathrm{P}(t - \tau + 1|\alpha) + \lambda_p \log p(n_i|n_j)\} \end{aligned} \tag{2.4}$$

where the times $\tau \in R(t, \alpha)$ are allowable starting times determined by the phone

α's minimum and maximum duration, $\mathrm{P}(t-\tau+1|\alpha)$ is the phoneme segment duration probability, $p(n_i|n_j)$ is the node transition pronunciation probability, and $\{\lambda_a, \lambda_d, \lambda_p\}$ are the weights for combining the acoustic, duration and pronunciation scores. $J^*(t, n_j)$ is the best combined score reaching node n_j at time t. The final value at the end $J^*(T-1, n_t^p)$ is the word's total combined score. For this work, different pronunciations are treated as being equally likely and thus the node transition probabilities $p(n_i|n_j)$ are left out of the total score for a word to avoid penalizing words with multiple pronunciations since we are taking the most likely pronunciation and not the sum over all pronunciations.

In systems which use an HMM for the acoustic model, the algorithm for obtaining a word score is similar to Equation 2.4. The main difference is that a word's triphone PPN is expanded so that each triphone arc is replaced by the topology of an HMM. The PPN nodes then include HMM states (which are the nodes within a triphone) and null nodes (which are the nodes between triphones). Then the maximization of Equation 2.4 is always taken over the previous HMM state n_i at the previous time frame $\tau = t-1$ for the next set of possible HMM states n_j at time t. Note that there is no concept of a triphone segment within the HMM framework so the duration score $\mathrm{P}(L|\alpha)$ is left out.

Word Sequence Scoring

The search space for the first stage of recognition is dependent on the number of words in the lexicon V, the type of language model that is used, the type of acoustic model that is used, the length of each utterance and the PPN topologies in the phonetic dictionary. For example, for an HMM acoustic model the search space at any given time is given by:

$$K_a + K_e V \tag{2.5}$$

where K_a is the number of active within-word nodes and K_e is the number of active end-word nodes as determined by the PPN and HMM topologies. The last factor, K_eV, is the number of unique words in context that will start at the next time frame. A node is "active" if it is currently in the search space, which for a beam search means that the probability is above some threshold. For an n-gram language model, Equation 2.5 potentially approaches a degree n polynomial as the search progresses in time. The search space for the SSM is bigger, since each arc must be scored over a set of begin and end times for all the within-word and cross-word arcs (the search space for rescoring with the SSM will be discussed in more detail in Section 4). The search objective with any acoustic model is to find the best scoring path forward in time through the search space from the beginning to the end of the utterance.

There are two basic types of search algorithms that have been used in one pass recognition systems: depth-first search algorithms and breadth-first search algorithms. The most common depth-first algorithm used for speech recognition is an approximate A^* stack search algorithm [7, 8]. This is a best-first search which saves all incomplete sentence hypotheses on a stack and extends the hypothesis with the highest estimated likelihood for the entire utterance. This search strategy requires estimating the likelihood of an incomplete hypothesis to the end of the utterance and combining this estimate with the partial score of the hypothesis to obtain an estimate of its overall likelihood.

By far the most prevalent search technique is the breadth-first "beam search". This is also known as time-synchronous Viterbi decoding with a beam threshold. The beam search is essentially a DP search using an equation similar to Equation 2.4 which is evaluated forward in time. If any state at a particular time has a likelihood that is below some fixed threshold (the "beam width") of the highest likelihood found for all states at this time, then that state is "deactivated" for this time. Thus low scoring hypotheses at a particular time are eliminated from the set of allowable paths through this point in the search space. The result of this search can be visualized as a tree that extends through the search space towards the end of the utterance with many branches "pruned" along the way.

For very large vocabulary speech recognition, the beam search requires a large amount of memory and computation in order to achieve good performance. The Cambridge University one pass decoder [9] uses special pruning and network construction techniques to overcome these problems. The decoder uses an algorithm similar to Viterbi decoding where all states that can be merged are collapsed into a tree.

2.2 Search Approaches – Multi-Pass

Large vocabulary speech recognition systems generally use two or more search stages. In the first stage, some kind of a "fast match" is used to limit the amount of searching that is done with computationally expensive knowledge sources (KS's). Figure 2.7 depicts two types of fast match stages: one where a fast match is used to limit the word hypotheses evaluated with the other KS's within the utterance, and one where the fast match produces a finite set of sentence hypotheses evaluated with the other KS's for the entire utterance. Any combinations of the two stages depicted in Figure 2.7 can be used to design the first recognition stage. The design of the first recognition stage is critical because it determines the complete set of hypotheses that will be passed on to later stages. The main focus of this work is in the last pass of a multi-pass recognition system so the fast match won't be discussed in this section.

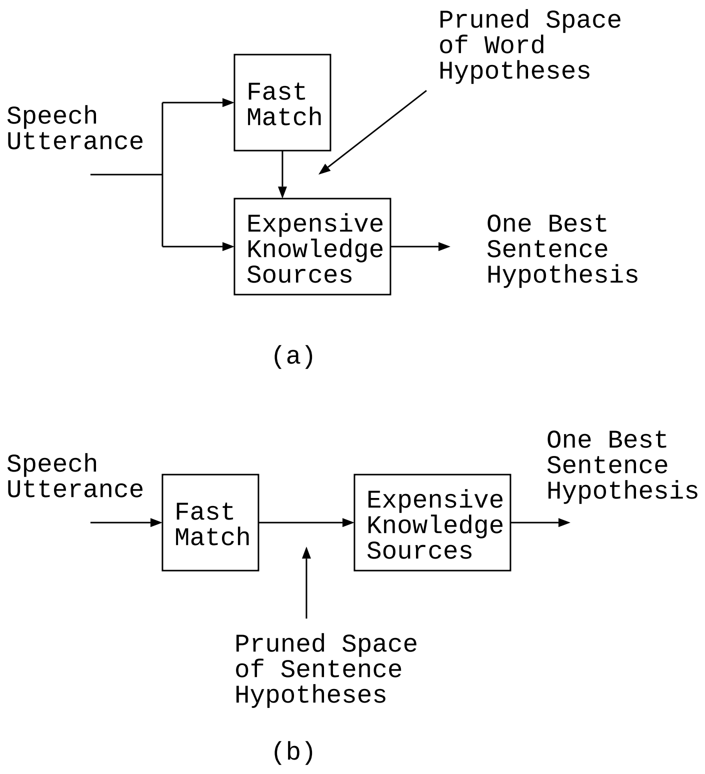

Pruned Space of Word Hypotheses
Speech Utterance
Fast Match
Expensive Knowledge Sources
One Best Sentence Hypothesis
(a)
Speech Utterance
Fast Match
Expensive Knowledge Sources
One Best Sentence Hypothesis
Pruned Space of Sentence Hypotheses
(b)

Figure 2.7: Search approaches using a fast match. (a) Throughout the sentence, the fast match limits the possible word hypotheses that are rescored at each point in time using more expensive knowledge sources. (b) A fast match produces a limited set of whole sentence hypotheses that are subsequently rescored using more expensive knowledge sources.

2.2.1 Fast Initial Search Stages

The forward-backward search [10] is a two-pass beam-search algorithm. Both passes use a beam search as described above, but they are initiated from opposite ends of the utterance. Also, the first pass may use less powerful and less computationally expensive acoustic and language models than the second pass. The second pass uses the scores generated by the first pass to estimate the likelihood of a partial hypothesis to the end of the utterance for pruning. A lower beam width can then be used in the second pass without increasing the number of decoding errors. Thus the search space is reduced for the more powerful models of the second pass. This algorithm has been shown to increase the overall decoding speed by 40 times over the standard beam search [10].

The forward-backward search algorithm gets its name from the Baum-Welch forward-backward training algorithm for HMM's. In the Baum-Welch algorithm, forward and backward likelihoods are calculated using a similar approach to the beam search. In contrast to the forward-backward search algorithm, the Baum-Welch algorithm does not require finding the most likely state sequence ending at a particular time. Instead, the algorithm only requires obtaining the forward-backward likelihood for being in an HMM state at a particular time. These likelihoods are then used for estimating the parameters of the HMM.

The above mentioned search techniques can be used as part of a strategy for generating lists of the most likely N hypotheses for an utterance. The N-best search paradigm was originally proposed as a means of integrating natural language processing in speech understanding or for subsequent rescoring by later recognition stages [11]. There is an exact N-best algorithm that uses a beam search described in [11]. In this algorithm, the N highest scoring unique partial sentence hypotheses ending at a particular time are saved for all times during the beam search. Also,

Figure 2.8: Performance vs. N for N-best search algorithms (from [12]).

probabilities for different paths with the same partial sentence hypothesis ending at a given time are summed, and a state-dependent threshold, which is higher than the global beam width, is used for pruning at the state level.

More efficient approximate N-best algorithms have been developed since the exact algorithm. These include the lattice and word-dependent N-best algorithms [12]. The lattice N-best algorithm saves a traceback reference at the beginning of each word for all entering words at each time within the utterance. Clearly this is much less information than is stored by the exact algorithm which saves the N unique sentence hypotheses reaching each state and time. The word-dependent algorithm saves separate theories for each state depending on the previous word. This algorithm is more expensive than the lattice algorithm, but it achieves better performance. Figure 2.8 shows a plot of the cumulative percentage of correct answers vs. the rank of the correct answer for the different N-best algorithms [12].

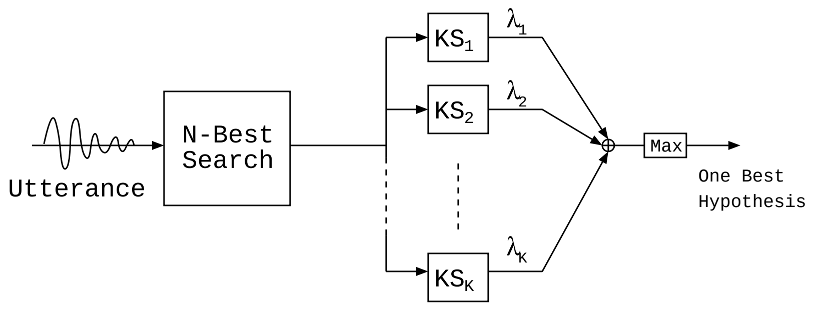


Figure 2.9: Schematic of the N-best paradigm.

2.2.2 The N-Best Rescoring Paradigm

One of the most prevalent approaches for incorporating two or more stages in a speech recognition system has been the "N-best search paradigm" [13]. Figure 2.9 is a diagram of an N-best recognition system. The first stages generate a list of N hypotheses for each utterance. This list is then passed on to other stages containing knowledge sources which are used to re-evaluate the list and produce a corresponding set of scores. At the end, the weighted combination of the scores obtained from each stage is used to determine the one best hypothesis. N-best rescoring has been used for speech recognition by BU [14], BBN [15], SRI [16], MIT and NYU [17].

A key issue for N-best systems is that the weights used for the score combination must be estimated. Weight estimation requires some kind of a design criterion, such as minimum word error rate, and an optimization scheme using a development set of utterances [13, 18]. The number of weights, or dimensionality of the weight vector, is an important consideration because there must be enough utterances and long enough N-best lists to estimate them reliably.

The BU recognition system [19, 14] uses the N-best paradigm with the SSM as one of the knowledge sources. The BBN BYBLOS HMM decoder [20] has been used to provide N-best lists of hypotheses. Knowledge sources used for the final score combination of each hypothesis currently include some or all of the following: the phone duration scores, the number of words, the number of phones, the number of silence phones, the SSM acoustic score, the BYBLOS HMM acoustic score, the BBN segmental neural net score, the BBN trigram language model score and the BU mixture LM score. The BU system searches for all allowable pronunciations of each word and also for optional silences between the words. The search space for the SSM is constrained to a time window around the phonetic segmentations from the BBN N-best decoder in order to reduce the order of complexity of the rescoring, as described in Section 3.

2.2.3 Lattice-Based Search Paradigms

Lattice-based search approaches have evolved as another means of incorporating two or more recognition passes into a speech recognition system. In some cases, N-best rescoring is still performed as the last recognition pass [20].

The term "lattice" has been used inconsistently in the speech recognition literature. In general, a lattice is a graph of words and word transitions that is used to represent a pruned search space, or a set of hypotheses generated by a speech recognition system. In some cases, a "lattice" is a finite-state grammar with no time information [20, 21]. In other cases, a "lattice" is an acyclic network of words with word transition time information [22]. In this work, we use the term "lattice" to mean an acyclic network of words with a unique start and end node annotated with time and score information. In addition, the term "N-best lattice" refers to lattices which are generated by the same algorithm which generates N-best hypotheses. The N-best lattices used in this work will be discussed in greater detail in Section 3. The term "word graph" will be used to mean an unannotated lattice (i.e. a finite-state word grammar).

SRI has used word graphs in a "progressive search" paradigm [21]. In this approach, the word graphs do not contain any time or score information but are used by latter stages as grammars to constrain the search space [21]. A similar approach has been used for the fourth pass of the BBN lattice decoder [20]. CMU has used lattices, where the score and timing information produced by the first stage of recognition is retained and used for score combination and decoding during the second recognition stage [23].

2.2.4 Local Search Algorithms

Local search algorithms have been used to solve NP-complete or NP-hard problems, where there is no known algorithm that will take less than exponential time for finding the exact solution [24]. For these problems, local search techniques have been used to formulate an approximate algorithm that is not guaranteed to converge at the globally optimal solution, but takes less than exponential time. There are several other approximate algorithms that have been applied to NP-complete problems, most notably simulated annealing [25] and genetic algorithms [26].

Local search algorithms are initiated at some feasible solution to an optimization problem. At each iteration, a local neighborhood of the current feasible solution is searched to find a better feasible solution. When no improvement can be found, the algorithm has converged.

A local search technique has been applied to speech recognition in the "split and merge" algorithm [27], where it is used to solve a phone classification and segmentation problem. For this problem, a polynomial-in-time algorithm is known, but computation is dominated by Gaussian segment score calculations. Local search techniques are used as a way of reducing the size of the search space for the algorithm and hence reducing the number of segment score evaluations.

A local search algorithm has been developed as part of this thesis work which searches lattices using the SSM to find the one-best answer. Like the phoneme recognition problem, this is a shortest path problem where the cost evaluation of each word uses the SSM. The SSM is a computationally expensive model, and the exact search space is quite large. Therefore, local search techniques have been used to limit the search space that is evaluated using the SSM, taking advantage of prior recognition search passes to provide a good initial path. The local search algorithm also provides a means of incorporating sentence-level knowledge sources into the search since there is a complete sentence hypothesis at each iteration of the search.

2.2.5 Summary of Different Search Approaches

In this section, the N-best search algorithm and various types of lattice-based search algorithms were presented. In general, all of these search approaches have some obvious deficiencies. The N-best algorithm has the problem of being expensive when N is large. The lattice-based search algorithms, on the other hand, are efficient, but do not allow for the incorporation of sentence-level knowledge sources into the search. The lattice local search algorithm which will be discussed in Chapter 5, however, has the potential advantage over N-best rescoring of being more efficient while also allowing for the incorporation of sentence-level knowledge sources into the search.

Chapter 3

Paradigm for Speech Recognition

This chapter gives an overview of the research paradigm that is used for evaluating the lattice DP, lattice local search and N-best search algorithms. The performance of these three algorithms was examined on two tasks: the ARPA Wall Street Journal dictation task and the Switchboard task defined at the Rutger's 1994 workshop in speech recognition. A detailed description is given of how N-best rescoring is performed by the BU recognition system and some important baseline results are presented. Also, details are provided about the modifications that were made to the BBN lattice decoder in order to generate N-best lattices that could be used with the BU lattice recognizer for running on the different tasks.

3.1 Corpora

The N-best, lattice DP and lattice local search algorithms have been evaluated on three different tests. The tests include the P0 condition of the 5,000 word Wall Street Journal Hub 2 test (WSJ-H2-P0), the P0 condition of the 20,000 word Wall Street Journal H1 test (WSJ-H1-P0) and the 5,000 word Switchboard test set defined at the 1994 Rutgers workshop in speech recognition [28]. The N-best search approach is used as both a baseline for speed and performance and as a means of estimating weights for combining knowledge sources for the two lattice-based search algorithms.

The corpora used in this work each consist of three sets of data: a training set, a development test set, and an evaluation test set. The training set consists of speech data for training acoustic models and textual data for training language models. The development test set is intended to be used for tuning parameters and developing the speech recognition system, and in our work is used for estimating weights for score combination. The evaluation test set is then used for occasional evaluations of the system. The design goal in all of the tasks is to minimize the over-all word error rate. The word error rate on a test set is defined as the total number of word insertions, deletions and substitutions that are made divided by the total number of words in the test set.

3.1.1 The ARPA WSJ Corpus

The purpose of the ARPA continuous speech recognition (CSR) evaluation is to sample the progress in speech recognition technology and provide a means of comparing different approaches to the CSR problem. The November 1993 ARPA evaluation was designed to use a "hub and spoke" paradigm [29]. This evaluation paradigm consists of two "hub" tests, the 5,000 word "H1" test and the 20,000 word "H2" test, and nine spoke tests. Each test has a specified number of optional and required conditions. The domain of all of these tests consists of Wall Street Journal (WSJ) news articles. All sites participating in the evaluation were required to run their systems on at least one of the hub tests.

The two WSJ hub tests consist of WSJ articles read by a variety of different speakers. The speech for both the training and testing data was collected using the same low-distortion microphone and noise-free recording environment. The allowable training data used for each test depends on the specific condition of the test.

The H1 hub has a required "C1" condition (H1-C1) and optional "P0" condition (H1-P0) and the H2 hub has a required "C1" condition (H2-C1) and an optional P0 condition (H2-P0). The acoustic training data and the language model are specified for the H1-C1, and H2-C1 conditions. Any acoustic training data and language model can be used for the H1-P0 and H2-P0 conditions.

The test data for both hubs consists of a development test set and an evaluation test set. Each development test set consists of five male and five female speakers with approximately fifty utterances per speaker. The evaluation test also has five male and five female speakers with approximately twenty utterances per speaker. The development data is intended to be used for tuning parameters in the recognition system and for performing research and development before running the system on the evaluation data. Typical word error rates for the 1993 H1-P0 condition are between 12.2% and 16.8%, and between 11.7% and 19.0% for the H1-C1 condition [30]. Typical error rates on the 1993 H2-P0 evaluation test set are between 4.9% and 9.2%.

3.1.2 The Switchboard Corpus

The Switchboard corpus consists recorded conversations over the phone about a variety of different topics. Unlike speech in the WSJ corpus, Switchboard speech is spontaneous and has many disfluencies. Also, some of the utterances have background noise such as a television or a baby crying.

During the 1994 Rutgers workshop in speech recognition, development and evaluation test sets were selected from the Switchboard corpus. Both test sets have five female and five male speakers with about 30 utterances per speaker. In order to generate the test sets, Switchboard telephone conversations where broken down into "turns" for each speaker where a "turn" corresponds to a speaker's turn to talk during a telephone conversation. The utterances for the test sets were selected by using full turns or phrases from the turns. Typical word error rates for both the development and evaluation test set are around 50%.

3.2 The BU N-best Rescoring Paradigm

The Boston University speech recognition system performs recognition by using the SSM to rescore hypotheses that have been produced by a first–pass recognition system. In some cases, the hypotheses are also rescored with the BU sentence-level mixture LM. For both of these knowledge sources, the search space is very large. The rescoring paradigm provides a means of incorporating expensive knowledge sources such as the SSM and long-distance knowledge sources such as the sentence-level mixture LM into the speech recognition search by providing a constrained search space for them. In this section, we describe the N-best rescoring search algorithm that was used prior to this thesis.

N-best rescoring with the SSM requires performing the DP iterations similar to Equation 2.4 over a triphone-expanded network for each hypothesis. The networks are constructed by concatenating the PPN's for each word in a hypothesis and adding optional silences between the words. Then the whole network is expanded to include triphone contexts. If no constraints are provided other than the minimum and maximum durations for each phone, then rescoring is expensive since each triphone in the network must be evaluated over a very large set of possible segmentations.

Let the set $\{n_s^t, n_t^t, N^t, A\}$ represent the triphone-expanded network with source

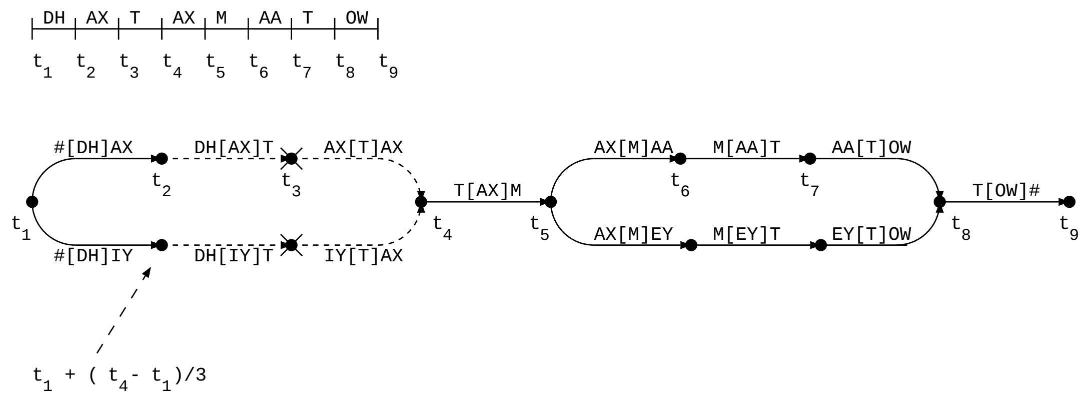


Figure 3.1: Marked triphone network with initial segmentation.

node n_s^t, destination node n_t^t, topologically sorted node set N^t and triphone arc set A. Also, let $R(n_i)$ be the time constraints associated with the node n_i. The algorithm for rescoring the triphone-expanded networks is given by:

1. Initialize: $J^*(0, n_s^t) = 0$

2. For each $n_j \in \{N^t - n_s^t\}$

   For $t \in R(n_j)$, sequentially calculate:

$$\begin{aligned} J^*(t, n_j) &= \max_{\tau \in R(n_i), \alpha_l[\alpha]\alpha_r(n_i,n_j) \in A} \{J^*(\tau, n_i) + \mathcal{L}(\alpha_l[\alpha]\alpha_r(n_i, n_j), \tau, t) \\ &\qquad + \log \mathrm{P}(t - \tau + 1|\alpha)\} \end{aligned} \tag{3.1}$$

In order to reduce the search space for rescoring hypotheses with the SSM, initial N-best phonetic segmentations of each hypothesis are used. Then the possible segmentations of each utterance can be constrained to be within $\pm n$ frames of the initial segmentation where n is typically 10. The nodes of the triphone-expanded network are then marked so that each node in the path of the initial segmentation has the corresponding segmentation time. Figure 3.1 depicts the time marked triphone-expanded network for the hypothesis "the tomato" with the corresponding initial segmentation. The nodes marked with an $\times$ are the transition nodes between the two words.

Once the network has been marked, the time constraints for the unmarked nodes are found. The unmarked nodes correspond to all of the pronunciations in the hypotheses' phone network that were not observed in the initial HMM segmentation. The constraints for the unmarked nodes were estimated by first finding all the unmarked paths between marked nodes. For each unmarked node in each path, a time is found by allowing all the unmarked nodes to be at equal time intervals between the marked node times. Then the time constraints $R(n)$ for an unmarked node n are represented by the minimum and maximum times which contain all estimated segmentation times for all paths through the node. In Figure 3.1 there is only one unmarked path between t_1 and t_4 and the first unmarked node of this path is marked with the time $t_1+(t_4-t_1)/3$. Once all of the nodes have been marked, the constraints for each node are relaxed by allowing an additional w_{DP} frames around each time or time window where w_{DP} is typically 10 frames.

The time constraints for each node in the network are used as constraints for the DP iterations of Equation 3.1. The knowledge sources used for rescoring include the SSM, a segment duration model, the number of phones and the number of inter-word silences. In the past, no other knowledge sources were combined at this stage. After the DP iterations are completed for a hypothesis, the SSM score, duration score, number of phones and number of silences for the rescored hypothesis are printed out to a file.

During DP, score caching is done to prevent redundant calculations, which can be significant for rescoring because many hypotheses share the same words. For the SSM, two types of score caches are used: a distribution score cache and a segment score cache. If a segment has already been scored, then its score is looked up in the segment score cache. If a segment has not been scored, then some of the frame scores within the segment may have been computed and stored in the distribution score cache.

Once all of the hypotheses have been rescored, the next step is to find the optimal weights for combining their scores. Weights are generally estimated using a held-out set of utterances, such as a development test set. Weight estimation is an unconstrained multi-dimensional optimization problem, which we solve using a grid-based optimizer which searches for the weights that re-rank the hypotheses to minimize the word error rate over the entire set. Then the system is evaluated on an independent set of utterances, such as an evaluation test set, using weights that are fixed to the values that were found using the grid-based optimizer.

3.3 Lattices

As part of this thesis work, the BBN decoder was modified in order to produce a lattices of word hypotheses for each utterance in a test set. In order to rescore these lattices quickly using the BU SSM, the lattices had to contain a phonetic segmentation for each word, as in N-best rescoring. Therefore, the BBN decoder was modified to save phonetic segmentations and to generate lattices.

The BBN decoder is a four pass recognition system. The first pass is a forward fast beam search using inexpensive (and inaccurate) models. The second pass is a backward beam search with five-state HMM's for each triphone, no cross-word triphones, and a bigram LM. Score information from the first pass is used for pruning in the backward pass. The third pass uses the same models as the second pass in a forward beam search. In the third pass, the second pass's backward scores are used for pruning.

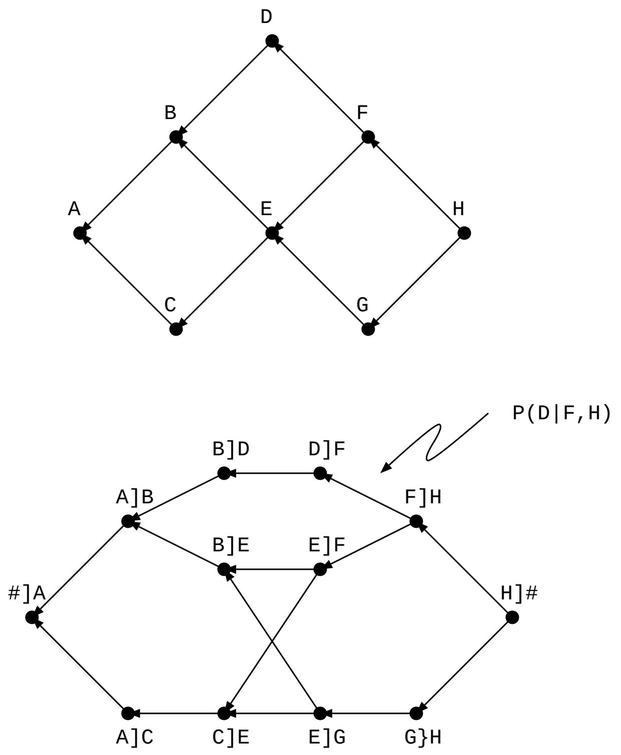


Figure 3.2: A word graph and its corresponding trigram expansion.

After the third pass, a graph is constructed using "observations" of word transitions from the second and third passes. A word transition is "observed" if the forward score reaching the end of a word plus the backward score reaching the beginning of the next word is above some threshold for a window of times. In the graph, words are represented as nodes with the transition between words represented as arcs. Arcs are added to the graph when new word transitions are observed, and nodes are added to the graph if new words are observed (a word is new if no transitions have been observed to the word for a given time window). After the graph has been fully constructed, it is expanded in the reverse direction so that each graph node has a unique right context. Figure 3.2 gives an example of a graph before and after the expansion. Each arc in the graph now has a unique backward trigram context and can be annotated with LM scores. For example, the arc D]F $\leftarrow$ F]H is annotated with the trigram language model probability $P(D|F, H)$.

The BBN decoder uses the backward trigram-expanded graph as a grammar for the fourth recognition pass. First the word nodes of the graph are expanded to their triphone phonetic pronunciations. Pronunciation networks are not used so words with multiple pronunciations must be duplicated in the graph. Then optional silences are added between words. Next, cross word triphones are added. A merge is done to reduce the size of the triphone graph by merging the last cross-word triphones of all word nodes that share the same right context in the graph. Now all words in the triphone-expanded graph have a single copy of their last cross word triphone for all cross-word transitions to the word.

The cross-word triphone merge is an approximation since it assumes that the segmentation of the last phone of a word is independent of the previous word in the backward word graph. However, the approximation is reasonable since each word in the graph has a unique right context. In most cases, the segmentations of the first

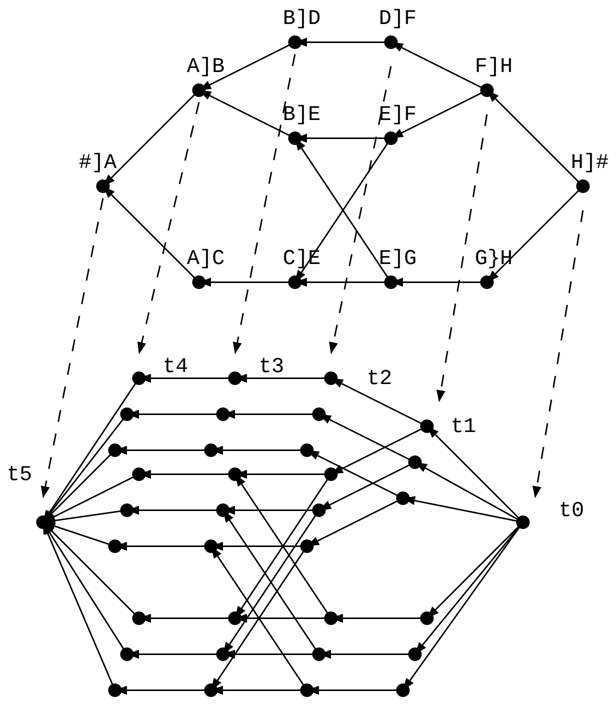
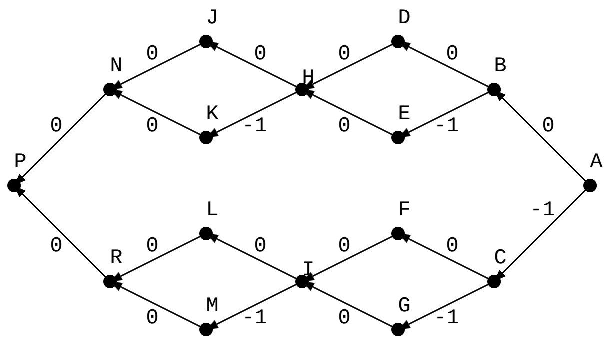


Figure 3.3: Example of a trigram-expanded word graph and the associated traceback after decoding. The parallel paths represent different traceback times for the same word sequence.

phone of the previous word nodes (which are all different copies of the same word) will be the same. The first phone of short words, such as one-phoneme words like "a", are more likely to have different segmentations.

The fourth recognition pass uses a five-state HMM with cross-word triphones and a trigram LM in a backward beam search using the expanded triphone graph. At each time during decoding, different theories coming in to a node where words merge are saved in a list along with their path scores. Each theory has a pointer back to the best previous list of theories. The resulting data structure of lists and pointers forms a traceback lattice which is associated with the original trigram-expanded graph in that each node of the lattice has a corresponding node in the graph. Figure 3.3 shows



```
A B D H J N P    0
A B E H J N P   -1
A B D H K N P   -1
A C F I L R P   -1
A B E H K N P   -2
```

Figure 3.4: The traceback structure after sorting and score normalizing and the corresponding N-best list for N=5 with scores.

an example of a trigram-expanded word graph and its associated traceback lattice. Each node of the traceback lattice is uniquely identified by a time and a node in the trigram-expanded graph.

In order to generate N-best hypotheses that are needed at least for weight estimation, each of the lists in the traceback lattice is sorted by score and only the highest scoring arc pointing back to a given word is saved. The scores in each list are then normalized by subtracting the highest score from all of the scores in the list. The optimal path reaching a node in the traceback lattice now has a score of zero and all other paths have negative scores. The N-best list is constructed by walking through the traceback lattice and accumulating the total normalized score for each path. The accumulated score for a path is the difference in score between that path and the best path through the traceback lattice. Thus the N hypotheses with the highest accumulated normalized scores are the N-best hypotheses. Figure 3.4 is an example of a lattice with normalized score, the corresponding N-best list for N=5 and

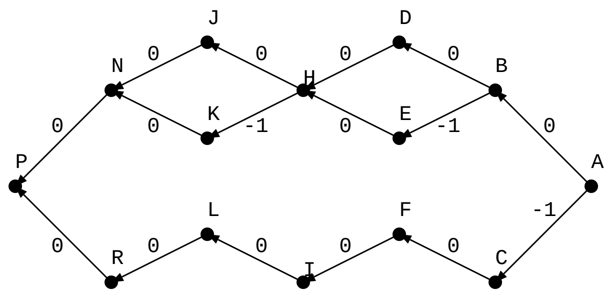


Figure 3.5: The N-best lattice corresponding the traceback lattice and N-best list of Figure 3.4.

the accumulated normalized scores for each hypothesis.

The word hypothesis lattices used for rescoring by the BU recognition system are sub-lattices of the traceback lattices produced by the BBN decoder. These lattices will be referred to as N-best lattices since they correspond directly to an N-best list. The N-best lattices are created by marking nodes in a traceback lattice which belong to paths in the corresponding N-best list and saving the sub-lattice which consists of the marked nodes. Figure 3.5 is the N-best lattice corresponding to the traceback lattice and N-best list of Figure 3.4.

During decoding, extra traceback structures are saved for each phone ending at a given time. These structures provide a means of obtaining the phonetic segmentation of each word in the N-best lattice. After decoding is completed, the N-best lattice is saved to a file. Note that the lattice is backwards with respect to time since the grammar used in the fourth pass of the BBN decoder is backward.

The lattices are annotated so that each lattice node has a phonetic segmentation and an HMM acoustic word score. The segmentations are shifted by one phone because of the merging that is done during the triphone graph building. The HMM scores are approximate since probabilities are summed in the decoder for each path, but the word scores (log probabilities) are calculated by subtracting the highest score at the end of a word from the score reaching the beginning of the word (going backwards)[1]. The word scores correspond to the segmentations for the words which are shifted by one phone, because of merging that is done inside the decoder to save computation and storage. The phone, silence and word insertion penalties and the weighted LM score are taken out of each word score, since new penalties will be estimated. Each arc in the N-best lattice is annotated with its corresponding trigram score.

The proposed standard lattice file format specification for use by the CSR community is included in Appendix A and it also can be used with forward lattices.

[1]Actually, the sum of the scores at the end of the word should be subtracted from the score reaching the beginning of the word.

Chapter 4

The Lattice DP Search Algorithm

The lattice DP algorithm is guaranteed to find the best scoring answer in a lattice given a set of Markov knowledge sources and an optimized set of weights. The search is more efficient than N-best rescoring because each word node in the lattice is only evaluated once for all of its phonetic segmentations. While triphone caching does improve the speed of N-best rescoring, it does not eliminate redundant computation above the segment level which can be substantial.

This chapter describes how triphone-expanded networks are constructed using the information that is in the lattices and then how time constraints are calculated for each node in the triphone-expanded network. A formulation is given of a DP algorithm that is performed backward in time (since the lattices are backwards) using the time constraints of the triphone-expanded network. Finally, weight optimization and N-best rescoring with the lattices is discussed.

4.1 Lattice Expansion

The lattice recognizer designed in this thesis reads in the lattices that are generated by the modified BBN decoder, described in Section 3.3, and constructs a triphone-expanded network which is used for rescoring with the SSM. The network is constructed by looking up each word associated with a lattice node in the dictionary and obtaining its PPN.

The PPN is modified to include a silence word[1] which is either long or short, end-utterance or inter-word silence, and may or may not be optional for this word. An end-utterance silence word is added if the node is at the beginning or end of the lattice. The difference between the two silence words is that the end-utterance silence word's PPN contains end-utterance silence phones while the inter-word silence word's PPN contains inter-word silence phones. Experiments on the Resource Management task showed that a different model for each type of silence improved performance of the recognition system.

The long silence words have paths for one, two or three silence phones in a row while the short silence words have only a single silence phone. The long silence words are used when more than n_s frames of silence are segmented by the HMM for the word, otherwise the short silence word is used. Also, if less than n_o frames of silence are observed, then the silence word is made optional in the modified PPN. This approach saves computation while allowing for some flexibility in the way that silences can be segmented. This silence representation is different from earlier BU rescoring work and the change was made to the lattice recognizer in order to save computation by restricting the possible silence segmentations in some places. For this work, n_s was

[1]In this thesis, we use the terms "silence word" and "silence phone" to indicate the acoustic model structure within the lattice. These are not "words" or "phones" in the normal sense of these terms.

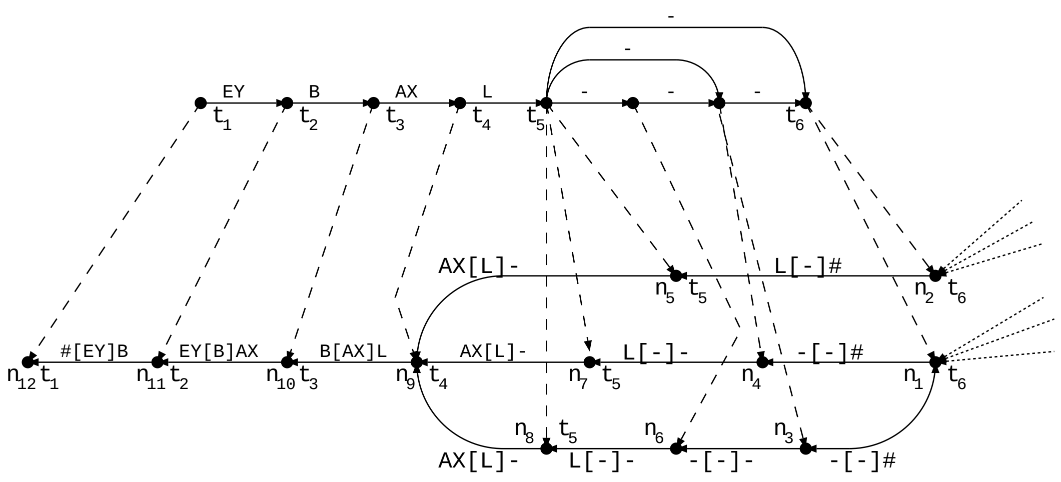


Figure 4.1: PPN for the word "ABLE" modified with a non-optional long inter-word silence ("-" is the inter-word silence phone). The corresponding triphone-expanded PPN is given below it.

set to 10 frames and n_o was set to 20 frames. In other words, an optional short silence word is used if there are less than 10 frames of silence, an optional long silence word is used if there are between 10 and 20 frames of silence and a non-optional long silence word is used if there are more than 20 frames of silence.

A triphone-expanded version of the modified PPN is created and a list is kept of all the nodes in the triphone-expanded PPN that correspond to a given node in the original PPN. Then the triphone-expanded PPN is added to the triphone-expanded network by attaching all of the cross-word triphone arcs. The newly added nodes are marked with segmentation times, and all nodes which correspond to the same node in the original PPN are marked with the same time. Figure 4.1 is an example of a modified PPN before and after the triphone expansion with pointers indicating the association between nodes in each network. The initial segmentation, $\{$(EY,t_1,t_2-1),(B,t_2,t_3-1),(AX,t_3,t_4-1),(L,t_4,t_5-1), (-,t_5,t_6-1)$\}$, was used to mark the modified PPN of Figure 4.1 and each associated node in the triphone-expanded network.

The time constraints for all of the unmarked nodes are calculated by enumerating all unmarked paths between marked nodes and using left-context mean durations. Specifically, the mean durations are used to determine how to proportion the unmarked node times between the marked node times. For example, in Figure 4.1 the unmarked paths are (n_1, n_4, n_7) and (n_1, n_3, n_6, n_8). Each paths goes from right to left since the triphone-expanded network is backward with respect to time. The time constraint for n_3, using left-context mean duration, is

$$T(n_3) = (t_6 - t_5)(\mu_{\mathrm{L[-}} + \mu_{-[-})/(\mu_{\mathrm{L[-}} + \mu_{-[-} + \mu_{-[-}).$$

If there had been more than one unmarked path going through n_3 then the time constraints for n_3 might have been a window with minimum and maximum constraint $T_{\min}(n_3)$ and $T_{\max}(n_3)$. After all the nodes have been marked, the time constraints $R(n)$ for each node n are further relaxed by allowing an extra λ_{DP} frame window around each set of constraints:

$$R(n) = \{T_{\min}(n) - \lambda_{DP}, \ldots, T_{\max}(n) + \lambda_{DP}\}.$$

For this work, λ_{DP} was set to 10 frames.

Figure 4.2 is an example of a lattice and its triphone expansion. Arrows indicate the triphone expanded modified PPN's in the network which correspond to words in the lattice. The dashed arcs are cross-word triphones and the $\times$ symbol indicates a word transition. Note that all of the nodes in the triphone-expanded network are either cross-word nodes (the ones that are marked with an "$\times$"), or within-word nodes. "</s>" is the used to represent the silence word. Note that each modified PPN, which may include a silence word, is considered to be a single word in this work.

Each triphone arc of the triphone-expanded network has an associated incremental graph score. The incremental graph score includes the weighted scores which

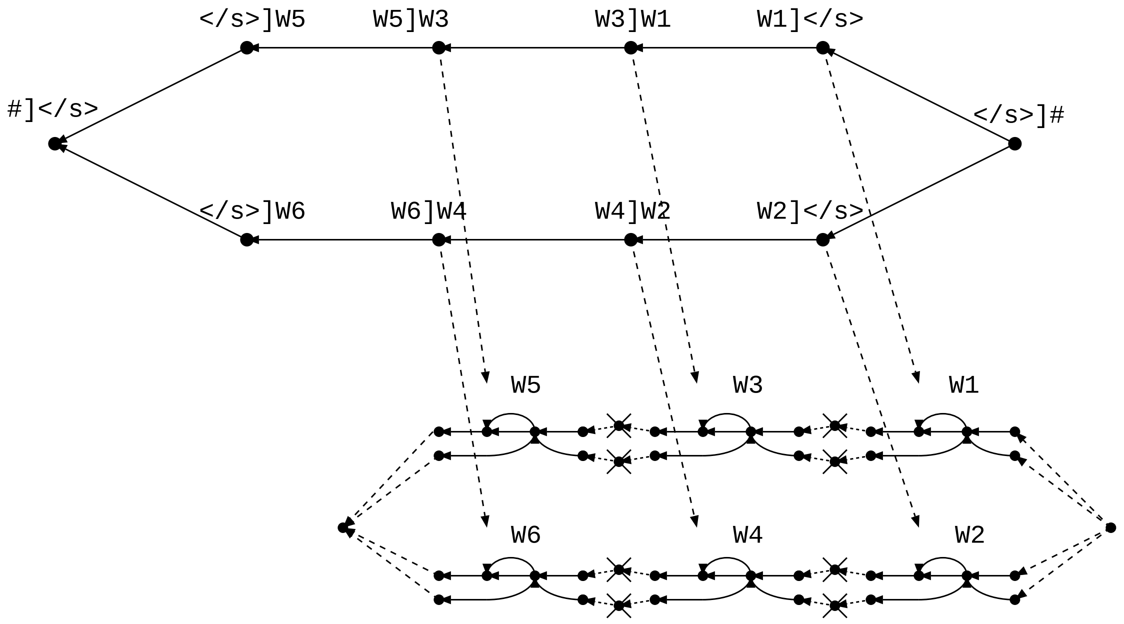


Figure 4.2: Lattice and triphone-expanded network with arrows indicating the PPN corresponding to each word, where "</s>" is the end-utterance silence word and "×" marks word junctures. Note that the end-utterance silence words are merged with preceding or following word's PPN.

are dependent on the topology of the triphone-expanded network alone such as the weighted LM score, the word insertion weight, the phone insertion weight and the inter-word silence insertion weight. The incremental graph score corresponds to the incremental combined score for traversing a triphone arc in the network from these knowledge sources. For example, the cross-word triphones at the end of a word have an incremental graph score that is the weighted LM score from the corresponding lattice arc plus the word insertion weight, and all of the triphone arcs with an inter-word silence as the middle context have a graph cost that is the silence insertion weight. The SSM and segment duration scores are not included in the incremental graph score because they depend on the possible segmentation times of the triphones in the network and not just the topology of the network. The incremental graph score is given by:

$$G(\alpha_l[\alpha]\alpha_r(n_i, n_j)) = \begin{cases} \lambda_P & n_i \text{ not } \times\text{-word node, } \alpha \neq \text{“-”} \\ \lambda_P + \lambda_S & n_i \text{ not } \times\text{-word node, } \alpha = \text{“-”} \\ \lambda_P + \lambda_W + \lambda_{LM} \log p_{LM} & n_i \text{ is } \times\text{-word node, } \alpha \neq \text{“-”} \\ \lambda_P + \lambda_S + \lambda_W + \lambda_{LM} \log p_{LM} & n_i \text{ is } \times\text{-word node, } \alpha = \text{“-”} \end{cases} \tag{4.1}$$

where λ_P is the phone insertion weight, λ_S is the inter-word silence phone insertion weight, λ_W is the word insertion weight and λ_{LM} is the trigram LM weight. Note that since silence words are only used for creating modified PPN's (by adding silence phones in various places), and are not considered to be words themselves, the word insertion weight does not penalize silences. The method for estimating the weights of Equation 4.1 is described in Section 4.3.

4.2 Scoring

The lattice DP algorithm is a straightforward extension of the word DP scoring algorithm. Let the triphone-expanded network corresponding to a lattice be represented by the set $\{n_s^T, n_t^T, N^T, A^T\}$ where n_s^T is the first node of the network (going backwards) n_t^T is the last node, N^T is the set of all the nodes and A^T is the set of all triphone arcs. N^T is topologically sorted so that all of the scores reaching a node at a given time have been evaluated before its successor nodes are reached. The lattice DP algorithm for an utterance of length T with observation sequence Y_0^{T-1} is:

1. Initialize: $J^*(0, n_s^T) = 0$
2. For each $n_i \in \{N^T - n_s^T\}$ calculate:

   For each $t \in R(n_i)$

$$\begin{aligned} J^*(t, n_i) &= \max_{\alpha_l[\alpha]\alpha_r(n_i,n_j)\in A^T, \tau\in R(n_j)} \{J^*(\tau, n_j) + \lambda_a \mathcal{L}(\alpha_l[\alpha]\alpha_r(n_i, n_j), t, \tau) \\ &\quad + \lambda_d \log \mathrm{P}(\tau - t + 1) + G(\alpha_l[\alpha]\alpha_r(n_i, n_j))\} \end{aligned} \tag{4.2}$$

The difference between Equation 2.4 and Equation 4.2 is that that the DP iterations span sequences of words instead of just one word. Also, using the definition in Section 4.1, here the incremental graph score G is included, and the time constraints $R(n)$ are used for both the source and destination node of each triphone arc.

During the iterations of the lattice DP algorithm, a traceback is saved for the best sequence of arcs and times reaching each node and time. When the algorithm has reached the beginning of the utterance (or end in a forward pass), the traceback contains the optimal segmentation through the triphone network. The highest scoring hypothesis is found by following the traceback and printing out the words corresponding to each lattice node.

The average complexity of the lattice DP algorithm depends on the average window width for the time constraints of the nodes in the triphone network. Generally, this window w_{ave} is close to $2w_{DP}$. The average complexity is then $|A|w_{ave}^2$ times the cost of an arc evaluation since each arc in A is scored over the minimum and maximum constraints for its source and target nodes. Using the SSM, the evaluation of each arc over a pair of times is dominated by the multivariate Gaussian computations. Therefore, all other factors, such as the length distribution calculations, are not included in this complexity estimate since they are relatively insignificant.

4.3 Lattice N-best Rescoring

In order to perform the lattice DP algorithm, score combination weights must be estimated using sentence hypothesis scores corresponding to the annotated lattice. In general, the SSM scores obtained this way may be different than the SSM scores obtained using some other initial segmentation such as the individually resegmented N-best hypotheses that were used in prior BU recognition systems, because the segmentation time windows may vary. Using the lattices for N-best rescoring is a means of producing scores that come from the same knowledge sources and constraints that are used in the lattice DP and lattice local search algorithms. Therefore, we next describe an algorithm for N-best rescoring on the lattices. This algorithm has the additional advantage of lower storage costs for representing the time and score information associated with the N-best hypotheses.

In the lattice N-best rescoring algorithm, each hypothesis in the N-best list is rescored by turning off all paths through the lattice except for the path corresponding to the hypothesis. Given a hypothesis $\overline{W} = (w_1, w_2., , , , w_k)$, and a lattice with node set N^L and arc set $A^L = \{a(n_p, n_q)|n_p, n_q \in N^L\}$, the path through the lattice corresponding to $\overline{W}$ is marked so that

$$\text{Flag}(a(n_p, n_q)) = \begin{cases} \text{TRUE} & \text{if } a(n_p, n_q) \in A^L \text{ and } n_p \text{ and } n_q \text{ are part of the path} \\ & \quad \text{corresponding to } \overline{W} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

Now let every arc of the triphone-expanded network have a reference to the corresponding arc in the lattice. The reference is **nil** unless the target node of the arc in the triphone-expanded network is a cross-word node:

$$\text{LattArc}(\alpha_l[\alpha]\alpha_r(n_i, n_j)) = \begin{cases} a(n_p, n_q) & \text{if } n_j \text{ is a } \times\text{-word node and } a(n_p, n_q) \\ & \quad \text{corresponds to } \alpha_l[\alpha]\alpha_r(n_i, n_j) \\ \textbf{nil} & \text{otherwise} \end{cases}$$

Here it is important to distinguish between the lattice arcs, which are the arcs of the original word lattice that have been read in from a file, and the triphone-expanded lattice arcs, which are the arcs of the lattice which has had all of its words expanded to triphones. During DP, references only exist to the triphone-expanded lattice arcs, and the flag for scoring words in the N-best hypotheses are on the original word lattice arcs. Therefore, it is important to first find out if a lattice arc corresponds to a given triphone-expanded lattice arc (i.e. if LattArc() does not return **nil** for the triphone-expanded arc) and then to check the flag corresponding to the lattice arc to see if the DP should proceed (i.e. if Flag() returns TRUE for the lattice arc). In the cases where a triphone-expanded arc corresponds to a lattice arc and Flag() returns FALSE for the lattice arc, the DP steps should be skipped for the following word in the triphone-expanded lattice. To assure that the the word will be skipped, $J^*(t, n_i)$ is defined as **MinPossScore** if node n_i has never been reached at time t during the DP. Then the maximization step of the DP algorithm is only performed for prior nodes and times which have a score greater than **MinPossScore** reaching them.

Now the DP algorithm of Equation 4.2 becomes:

```
1. Initialize: $J^*(0, n_s^t) = 0$

2. For each $n_i \in \{N^t - n_s^t\}$ calculate:
      For each $t \in R(n_i)$
         if (LattArc($\alpha_l[\alpha]\alpha_r(n_i, n_j)$) $\neq$ nil and !Flag(LattArc($\alpha_l[\alpha]\alpha_r(n_i, n_j)$)))
            continue
         otherwise
```

$$J^*(t, n_i) = \max_{\alpha_l[\alpha]\alpha_r(n_i,n_j)\in A^T, \tau\in R(n_j), J^*(\tau,n_j) > \mathbf{MinPossScore}} \{J^*(\tau, n_j) + \lambda_a \mathcal{L}(\alpha_l[\alpha]\alpha_r(n_i, n_j), t,$$
$$+ \lambda_d \log \mathrm{P}(\tau - t + 1) + G(\alpha_l[\alpha]\alpha_r(\tag{4.3}$$

Equation 4.3 will be referred through out this thesis as the lattice N-best rescoring algorithm[2].

Weights are estimated for all the lattice search algorithms by first performing N-best rescoring using the initial segmentations and N-best lists corresponding to the lattices. After all the hypothese have been rescored, weights for combining the different knowledge sources are optimized using the procedure described in Section 3.2.

[2]The lattice N-best *rescoring* algorithm should not be confused with the lattice N-best algorithm which is used to generating N-best hypotheses.

Chapter 5

The Lattice Local Search Algorithm

The lattice local search algorithm, unlike the unpruned lattice DP algorithm, is a sub-optimal search algorithm. That is, the lattice local search does not guarantee that the optimal solution will be found given the knowledge sources that are used for the search. However, the lattice local search does allow for the incorporation of sentence-level or long-distance knowledge sources which cannot be used in the lattice DP algorithm. Consequently, the lattice local search has the potential of achieving higher accuracy than an optimal search which uses less powerful knowledge sources.

This chapter presents the details of the lattice local search algorithm. First, the general approach is outlined and then the local neighborhood and a local path are defined. Next, the evaluation criterion for choosing the best local path is described along with the approach taken for scoring different local paths in the local neighborhood. Finally, word score caching and the implementation of sentence-level knowledge sources into the local search algorithm are discussed.

5.1 General Approach

The local search algorithm is an iterative algorithm which is initialized at some solution to a search problem. At each iteration, the local neighborhood of the current solution to the problem is evaluated and the best choice in this local neighborhood becomes the new current solution. Iterations of the algorithm continue until the local neighborhood of the current solution does not contain a better solution to the problem. The local search algorithm consists of the following steps:

```
1. Define:
   $P_0$ is the initial path
   $P_{\text{cur}}$ is the current path
   $F(P)$ is the relative cost of path $P$
   $\mathcal{N}_{\text{local}}(P)$ is the local neighborhood of $P$
   $C_{\text{cur}}$ is the relative cost of the current path

2. Initialize:
      $P_{\text{cur}} = P_0$, $C_{\text{cur}} = F(P_{\text{cur}})$

3. Iterate:
      $C_{\text{best}} = \text{inf}$, $P_{\text{best}} = \text{nil}$
      For all $P \in \mathcal{N}_{\text{local}}(P_{\text{cur}})$
            If $(F(P) < C_{\text{cur}}$ and $F(P) < C_{\text{best}})$ then
                  $C_{\text{best}} = F(P)$
                  $P_{\text{best}} = P$
      if $(P_{\text{best}} == \text{nil})$ then stop
      $P_{\text{cur}} = P_{\text{best}}$
      $C_{\text{cur}} = C_{\text{best}}$
      Go to 3
```

Note that unlike the DP algorithm, at each iteration of the local search algorithm an entire solution to the search problem is known.

It is shown in [27] that, for the shortest path problem on an acyclic directed graph, the smallest neighborhood that is guaranteed to contain the globally optimal solution, or the "minimal exact" neighborhood, is the neighborhood which consists of all paths that "form one loop" with P_{cur}, where a path forms a loop with P_{cur} if it has two nodes in common with P_{cur} but no arcs in common with it.

For this work, the local search algorithm is performed on an N-best lattice which is an acyclic, directed graph. Also, $F(P)$ is the scoring function described in Section 3.2. The lattice local search algorithm is initialized with the best hypothesis found by the first pass recognition system. After the N-best lattice has been read in from a file, the triphone network has been created and the time constraints for each node of the triphone network have been calculated, all paths through the triphone network are "turned-off" except for the paths corresponding to the initial hypothesis. Equation 4.3, the lattice N-best algorithm, is used to find the best score and segmentation of the initial hypothesis (step 2, above). Then all local paths in the local neighborhood of the initial hypothesis are evaluated to find the one which increases the score of the initial hypothesis the most (step 3, above). Now a new hypothesis is constructed by adding this local path to the initial hypothesis and removing the part of the initial hypothesis which forms one loop with the local path. The new hypothesis becomes the initial hypothesis for the next iteration of the local search.

Clearly, if the lattice local search algorithm evaluates the entire minimal exact neighborhood to find the highest scoring hypothesis, it would be inefficient compared to DP. The goal in this work is to design a local neighborhood which can be searched efficiently and that also yields the globally optimal solution in a few steps most of the time. The design of the local neighborhood used in this work will be more fully

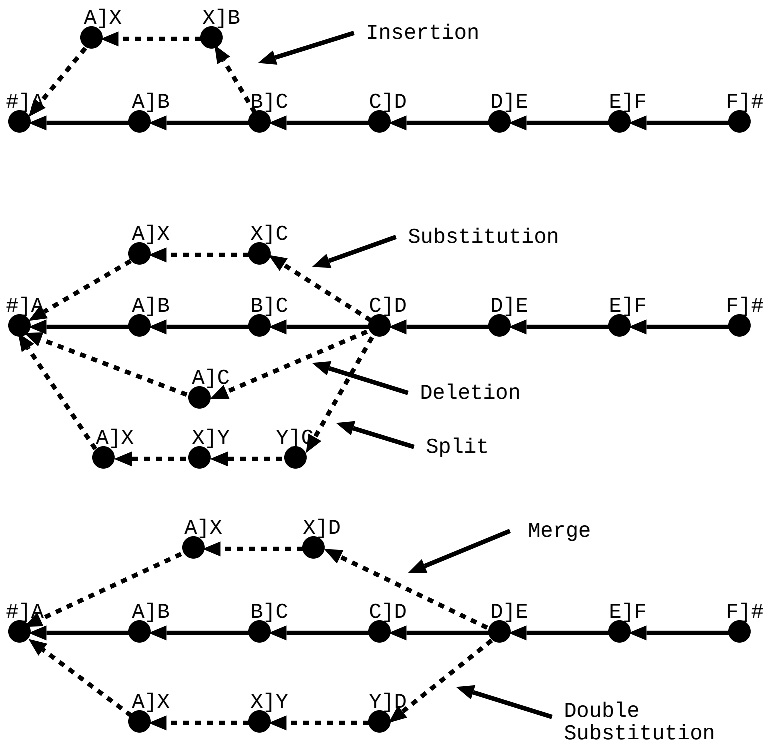


Figure 5.1: The six types of local paths that can be part of the local neighborhood for the lattice local search algorithm.

discussed in the next section.

5.2 Local Neighborhoods

The local neighborhood used for this work was designed to include loops with P_{cur} that correspond to the most frequent types of errors that are made in the N-best list. Most of the time, the hypotheses in the N-best list only differ by one or two words. Therefore, the loops included in $\mathcal{N}_{\text{local}}$ were selected so that they correspond to short loops of up to five nodes which are designed to bypass these types of errors.

The local neighborhood used for the lattice local search algorithm is analogous

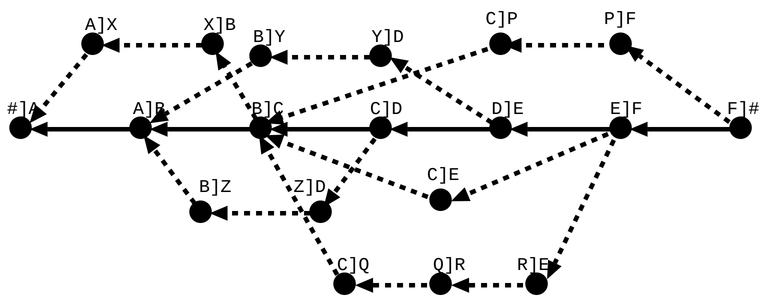


Figure 5.2: Local neighborhood of some initial path through an N-best lattice. Specifically, the neighborhood is the **set** of local paths where each dotted path in the figure is a unique local path. In this case, there are six local paths in the local neighborhood.

to the local search neighborhood used in the split-and-merge algorithm [27]. For the lattice local search algorithm, the local neighborhood is defined by six types of local paths: "insertions," "deletions," "substitutions," "splits," "merges," and "double substitutions." Each local path forms one loop with an initial path through the lattice. For the trigram-expanded initial path "#]A A]B B]C C]D D]E" (from "# A B C D E") an insertion replaces "A]B" with "A]X X]B" ("X" is inserted between "A" and "B"), a deletion replaces "A]B B]C" with "A]C" ("B" is deleted), a substitution replaces "A]B B]C" with "A]X X]C" ("X" is substituted for "B"), a split replaces "A]B B]C" with "A]X X]Y Y]C" ("B" is split into "X" and "Y"), a merge replaces "A]B B]C C]D" with "A]X X]D" ("B" and "C" are merged into "X") and a double substitution replaces "A]B B]C C]D" with "A]X X]Y Y]D" ("X" and "Y" are substituted for "A" and "B"). The local paths associated with each of these actions are depicted in Figure 5.1. Note that the set of local paths for some initial path through an N-best lattice depends on the topology of the N-best lattice. Figure 5.2 is an example of some initial path and the corresponding local paths in the local neighborhood.

Table 5.1: *Difference in 1-step inclusion rate and n-step coverage for two different local neighborhoods.*

Neighborhood	1-Step Inclusion Rate	n-Step Coverage
6 local paths	38.9%	92.8%
3 local paths	35.9%	90.2%

Two local neighborhoods were experimented with in this work. The first local neighborhood contained only three types of local paths: a "substitution," a "deletion," and a "split". This design decision was motivated by similarity between these local paths and the possible moves that are used in the split-and-merge algorithm [27]. The second type of local neighborhood consists of all six types of local paths described above. Table 5.1 gives the 1-step inclusion rate and n-step coverage for the two local neighborhood definitions. The 1-step inclusion rate is the percentage of time the correct hypothesis is in the local neighborhood of the HMM top hypothesis that is used to initiate the search. The n-step coverage is the minimum overall error rate that can be attained if the local path that reduces the number of word errors the most is picked at each iteration.

5.3 Local Path Evaluation

In order to score local paths, it is important to keep track of where the local path begins and ends in the triphone-expanded network. Also, the traceback for the current hypothesis must not be changed until the the best local path is found. The approach taken here is to use the same traceback structure for scoring all local paths in the local neighborhood. Therefore, the points of intersection between each local path and the current hypothesis must be kept track of. These intersections are referred to as "common nodes," and Section 5.3.1 discusses the role common nodes play in specifying local paths. Section 5.3.2 then outlines the algorithm for scoring the local path and updating the traceback structure after he best path has been found.

5.3.1 Common Nodes

For each triphone-expanded, modified PPN in the lattice, there are a set of unique nodes which are the first and last nodes common to all paths through the corresponding word node in the lattice. These nodes are before and after the cross-word triphone nodes respectively. They are called "common nodes".

For each word node in the lattice, two sets of common nodes are maintained: the set of all the common nodes which are the source of all triphone paths through the corresponding triphone-expanded PPN and the set of all nodes which are the destination of these paths. These sets are, respectively, the "source common nodes" and "destination common nodes" of the triphone-expanded, modified PPN corresponding to that word node in the lattice. Figure 5.3 is the triphone-expanded PPN of Figure 4.1. The word transitions nodes are marked with an $\times$ and the cross-word triphone arcs are the dashed lines.

At each iteration of the local search, an estimate is found of the maximum amount each local path can increase the score of the initial hypothesis for that iteration. Each local path is scored from the point where it branches off from the initial hypothesis to the point where it again meets the initial hypothesis. The nodes of the triphone-expanded network which correspond to the points where a local path branches off from the initial hypothesis are called the "takeoff nodes" of the local path and the nodes which correspond to the points where a local path joins the initial hypothesis are called the "landing nodes" of the local path. Figure 5.4 is an example of a local path

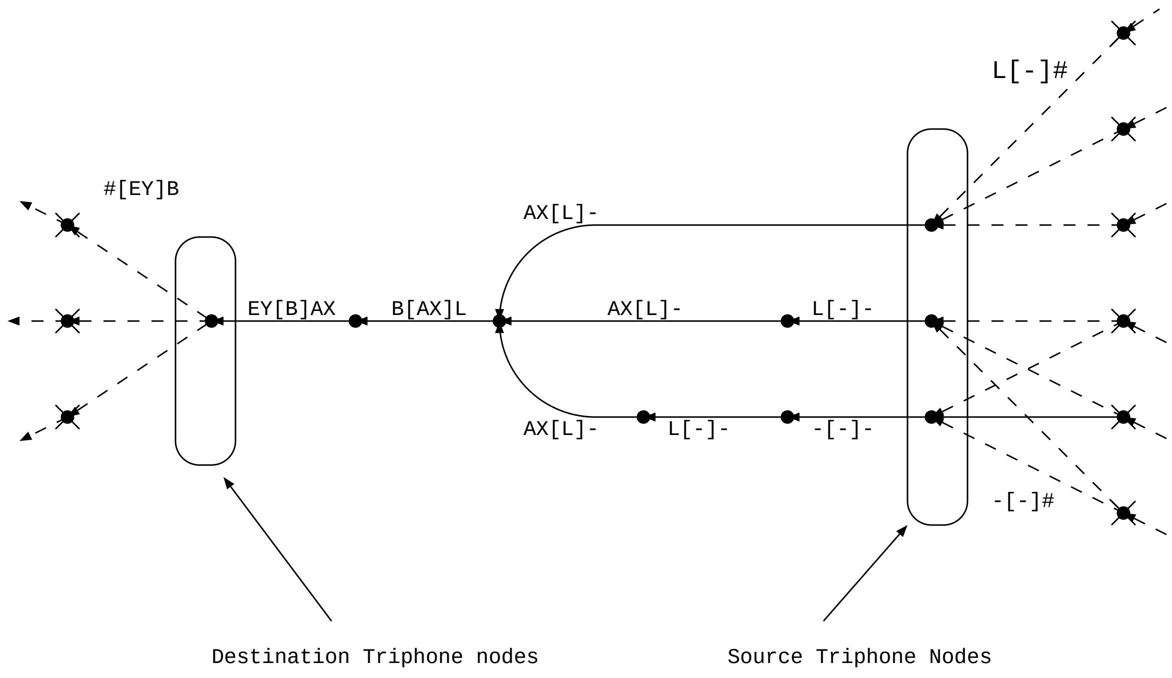
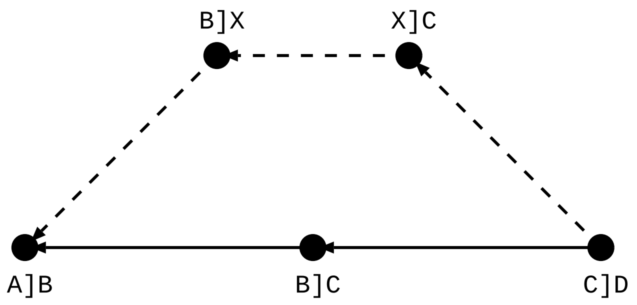


Figure 5.3: The triphone-expanded, modified PPN of Figure 4.1 with cross-word triphones in dashed lines. The word transition nodes are marked with an $\times$. The source and destination nodes of the PPN are also indicated, assuming a backwards lattice.



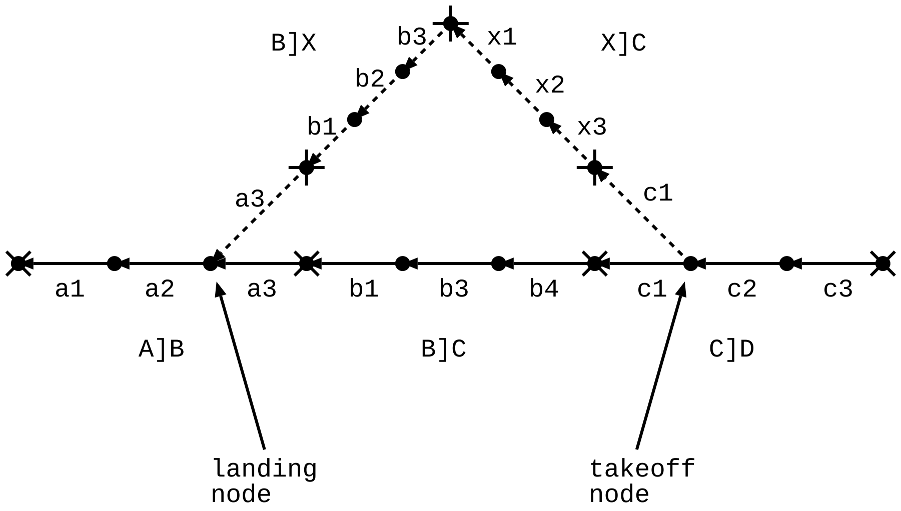


Figure 5.4: The local path corresponding to an insertion and the corresponding triphone-expanded network. To simplify the illustration, all words only have one pronunciation with three phones and triphone context identifiers have been left out.

corresponding to an “insertion” and the corresponding triphone-expanded network’s takeoff and landing nodes, for a backwards lattice.

A local path’s takeoff nodes are the source nodes of the first word of the path while its landing nodes are the destination nodes of the last word of the path. Under some circumstances, the first or last word in a local path can be “skipped”. This happens if there is a path through the word’s triphone-expanded, modified PPN which does not pass through any of the source or destination nodes of the PPN. For example, the triphone-expanded, modified PPN’s for the one-phoneme words such as “A” or “I” would be skipped if they did not include a non-optional silence because there would be a path through the PPN which consists of a single triphone and this would be in the cross-word set of phones that is outside of the standard takeoff and landing node definitions. To avoid “skipped” words, takeoff and landing nodes are moved outside the local path. Specifically, if the first word of a local path can be skipped, then the source nodes of the previous word in the current hypothesis are designated as takeoff nodes for the path. If the last word of a local path can be skipped, then destination nodes of the next word in the current hypothesis are designated landing nodes for the path.

5.3.2 Local Path Scoring

Three steps are involved in evaluating the local paths in the local neighborhood of the current solution. In the first step, the local path score must be found using DP. Then the second step involves caching the local path score so that it can be looked up later since different local neighborhoods can have the same local path. Finally, the third step involves calculating the increase in score for the local path relative to the current hypothesis.

After the initial hypothesis has been rescored, the traceback structure is modified so that each common node, at each time within its time constraints, points back to the best previous common node and time. Thus the traceback structure skips over all nodes that are not common nodes. This makes it possible to rescore local paths without touching the traceback for the initial hypothesis since the local paths will only access the initial hypothesis' traceback at landing or takeoff nodes.

Let the nodes $\{n_i^L, ..., n_f^L\}$ be the set of word nodes belonging to the local path P_{local} of an initial hypothesis where n_i^L is the initial word node of the path and n_f^L is the final word node of the path. Let **PathNodes** be a topologically sorted set of all nodes of the triphone-expanded N-best lattice which are contained in all pronunciations along the local path from its takeoff to its landing nodes including the takeoff and landing nodes themselves. The traceback data structure, **Trace**(t, n), is used to store a reference to the best prior triphone node and time reaching node n at time t and the corresponding best path score within the local path. The cache for the local path, **PathCache**(P_{local}), is used to store an array of all the best takeoff nodes and times for each landing node and time and the difference in score between the two.

Now the first step DP algorithm for scoring a local path using a backward lattice is given by:

Step 1:

```
for each $n_s \in \textbf{PathNodes}(P_{\text{local}})$
    for each $\alpha_l[\alpha]\alpha_r(n_d, n_s) \in A$ s.t. $n_d \in \textbf{PathNodes}$
        for each $t_i \in R(n_d)$
            for each $t_f \in R(n_s)$
                $\textbf{score} = \lambda_a \mathcal{L}(\alpha_l[\alpha]\alpha_r(n_d, n_s), t_i, t_f) + \lambda_d \log \mathrm{P}(t_f - t_i + 1) +$
                    $G(\alpha_l[\alpha]\alpha_r(n_d, n_s)) + \textbf{Trace}(t_f, n_s).\textbf{score}$
                if $(n_d \in \textbf{LandingNodes}(P_{\text{local}})$
        if ($\textbf{PathCache}(P_{\text{local}}, n_d, t_i)$.score < score)
            $\textbf{PathCache}(P_{\text{local}}, n_d, t_i)$.score = score
            $\textbf{PathCache}(P_{\text{local}}, n_d, t_i)$.pre_node = $n_s$
            $\textbf{PathCache}(P_{\text{local}}, n_d, t_i)$.pre_time = $t_f$
      else if ($\textbf{Trace}(t_i, n_d)$.score < score)
        $\textbf{Trace}(t_i, n_d)$.score = score
        $\textbf{Trace}(t_i, n_d)$.pre_node = $n_s$
        $\textbf{Trace}(t_i, n_d)$.pre_time = $t_f$
```

where G is given by Equation 4.1. Note that, since **PathNodes** is sorted topologically, the takeoff nodes are the first nodes in the set and the landing nodes are the last nodes in the set. Also, note that the traceback is not touched at the landing nodes of the path and so the traceback of the current solution is not effected by this operation. The above algorithm assumes segmental acoustic models and a trigram LM are being used. Local path evaluations with a sentence-level LM will be discussed in Section 5.5.

Now **PathCache** must be updated so that there is a reference back to the best takeoff node and time for each landing node and time, and the difference in score between these nodes and times is stored. This is essentially a traceback operation for all landing nodes and their associated times:

Step 2:

```
  for each $n_d \in \textbf{LandingNodes}(P_{\text{local}})$
    for each $t_i \in R(n_d)$
      $n_s = \textbf{PathCache}(P_{\text{local}}, t_i, n_d)$.pre_node
      $t_f = \textbf{PathCache}(P_{\text{local}}, t_i, n_d)$.pre_time
      while ($n_s \notin \textbf{TakeoffNodes}(P_{\text{local}})$)
        $n = \textbf{Trace}(t_f, n_s)$.pre_node
        $t_f = \textbf{Trace}(t_f, n_s).\textbf{pre\_time}$
        $n_s = n$
    $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_time} = t_f$
    $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_node} = n_s$
    $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{path\_score} =$
        $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{score} - \textbf{Trace}(t_f, n_s).\textbf{score}$
```

After the **PathCache** has been brought fully up to date, the traceback corresponding to all nodes in **PathNodes**, except for the takeoff and landing nodes, is reset so that it can be used again for scoring a different path. Note that the best segmentation is not available after this algorithm has completed, but it is also not required by the algorithm.

A paths's "delta score" is the maximum estimated amount that the path can increase the score of the current hypothesis. A path's delta score is found using the following algorithm:

Step 3:

```
$\textbf{max\_delta\_score} = \textbf{MinPossScore}$
for each $n_d \in \textbf{LandingNodes}(P_{\text{local}})$
    for each $t_i \in R(n_d)$
        $n_s = \textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_node}$
        $t_f = \textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_time}$
        $\textbf{path\_delta\_score} =$
            $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{path\_score} -$
            $(\textbf{Trace}(t_i, n_d).\textbf{score} - \textbf{Trace}(t_f, n_s).\textbf{score})$
        if ($\textbf{path\_delta\_score} > \textbf{max\_delta\_score}$)
            $\textbf{max\_delta\_score} = \textbf{path\_delta\_score}$
```

Note that the maximum delta score is found by finding the best takeoff node and time for a each landing node and time and comparing the difference in score between the current hypothesis and the local path between these two points.

After all delta scores of all paths in the local neighborhood of the current hypothesis have been found, the local path with the highest delta score is used to create the hypothesis for the next iteration. The new hypothesis is created by substituting the local path for the nodes in the current hypothesis that they form one loop with. Now the new hypothesis is rescored by starting at the takeoff nodes of the local path. This must be done because the traceback corresponding to the local path has already been reset and it must be filled in with the proper values. Also, the traceback times and scores beyond the local path's landing nodes must be updated, which involves resegmenting the rest of the utterance.

An attempt was made to try to reduce the amount of DP used to rescore each new hypothesis. The approach was to increment the scores in the traceback by the delta score for the best local path. This approach was found to hurt performance. The performance degradation may have been caused by the difference in segmentation for short words reaching the landing nodes of the path. For instance, one or two phoneme words may have very different segmentations if the previous words end at very different times. One approach for overcoming this problem would be to extend the scoring of the local path by a few phones into the current hypothesis beyond the landing nodes. After a few phones, the two separate paths should have the same segmentations.

5.4 Word Caching

Both the N-best and local search algorithm are plagued with the problem of redundant calculations. The problem is that each word node in the N-best lattices is

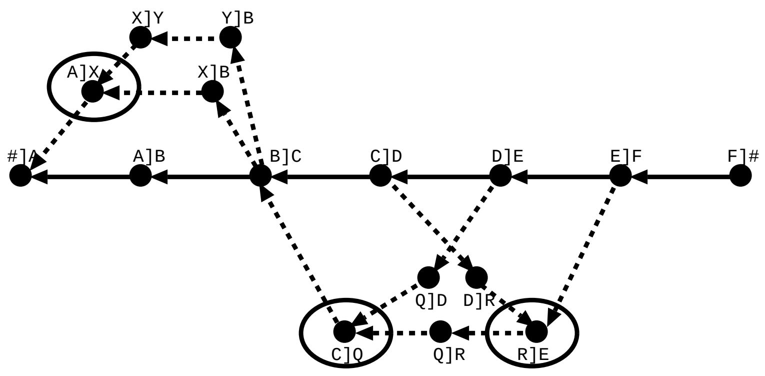


Figure 5.5: Example of local paths which share the same nodes in the lattice, where shared nodes are circled.

rescored many times by both algorithms. For instance, in the local search algorithm, many local paths share the same word nodes and they must be rescored separately for different paths. An example of local paths which share word nodes is given in Figure 5.5. This problem of redundant word node scoring can be alleviated through word score caching.

Since each word node in the N-best lattice has a unique right context, it is reasonable to assume that the segmentation and score of a word node is independent of the particular arc coming into that node. Word caching can be accomplished be keeping track of the best start time and destination triphone node for a given end time and source triphone node. If a word node has been cached, then DP is performed until all of the destination triphone nodes of the word node have been reached. Then the traceback is updated by adding each of the cached word scores associated with each of the source triphone nodes and end times to the score reaching the corresponding destination triphone node and start time in the traceback.

During DP along the triphone-expanded N-best lattice in **Step 2**, a record is kept of how many destination nodes have been reached for each word node of the original lattice. When all of the destination nodes have been reached for a particular word node, then the word is cached. Caching is performed by saving the best source node and time for each destination node and time. Let **SourceNodes**(n^L) be the set of all source nodes and **DestNodes**(n^L) be the set of destination nodes for the word node n^L of the N-best lattice. The traceback for the best source node and time for a given destination node and time is stored in **WordCache**(n^L).**trace**. The algorithm for caching a word once all of the destination nodes have been reached is as follows:

```
for each $n_d \in \mathbf{DestNodes}(n^L)$
    for each $t_i \in R(n_d)$
        $n_s = \mathbf{Trace}(n_d, t_i).\mathbf{pre\_node}$
        $t_f = \mathbf{Trace}(n_d, t_i).\mathbf{pre\_time}$
        while $(n_s \notin \mathbf{SourceNodes}(n^L))$
            $n = \mathbf{Trace}(n_s, t_f).\mathbf{pre\_node}$
            $t_f = \mathbf{Trace}(n_s, t_f).\mathbf{pre\_time}$
            $n_s = n$
        $\mathbf{WordCache}(n^L).\mathbf{trace}(n_d, t_i).\mathbf{score} =$
            $\mathbf{Trace}(n_d, t_i).\mathbf{score} - \mathbf{Trace}(n_s, t_f).\mathbf{score}$
        $\mathbf{WordCache}(n^L).\mathbf{trace}(n_d, t_i).\mathbf{pre\_time} = t_f$
        $\mathbf{WordCache}(n^L).\mathbf{trace}(n_d, t_i).\mathbf{pre\_node} = n_s$
```

If a word has been cached, and all of its source nodes have been reached during DP, then the traceback is updated using the cached scores and DP resumes after the first destination node (all of the nodes between the source and destination nodes for that word are skipped). This requires modifying **Step 2** with the following algorithm. Assuming all of the source nodes have been reached, the algorithm is as follows:

```
for each $n_d \in \mathbf{DestNodes}(n^L)$
    for each $t_i \in R(n_d)$
```

```
$n_s = \textbf{WordCache}(n^L).\textbf{trace}(n_d, t_i).\textbf{pre\_node}$
$t_f = \textbf{WordCache}(n^L).\textbf{trace}(n_d, t_i).\textbf{pre\_time}$
$\textbf{Trace}(n_d, t_i).\textbf{score} =$
    $\textbf{WordCache}(n^L).\textbf{trace}(n_d, t_i).\textbf{score}+$
    $\textbf{Trace}(n_s, t_f).\textbf{score}$
$\textbf{Trace}(n_d, t_i).\textbf{pre\_time} = t_f$
$\textbf{Trace}(n_d, t_i).\textbf{pre\_node} = n_s$
```

Table 5.2: *Relative speed with and without word caching for the lattice N-best and lattice local search algorithms. The speed measure is defined in Section 3.2.*

	Lattice N-best	Lattice Local Search
No Caching	87.4	8.6
Caching	83.4	8.7

Table 5.2 gives the relative speeds of the lattice N-best and lattice local search algorithms. The speed measurements are described in Section 6.1. Word caching seems to help only a small amount for N-best rescoring and does not help at all for the lattice local search. This could be because of the computation that is required to save and look up the cached scores. In addition, word score caching is only useful for Markov knowledge sources.

5.5 Sentence–Level Knowledge Sources

Since an entire sentence hypothesis is known at each iteration of the lattice local search, sentence-level knowledge sources can easily be incorporated into it. The lattice local search has this advantage over the lattice DP algorithm. Also, the local search algorithm is more efficient than N-best rescoring which is also commonly used for searching with sentence-level knowledge sources.

If sentence-level knowledge sources are used in the lattice local search algorithm, then **Step 3** must be changed to include the difference in the sentence-level score between the current hypothesis and the new hypothesis created by modifying the current hypothesis with a local path. Specifically, for a current hypothesis P_{cur}, new hypothesis P_{new} and a sentence-level knowledge source **SentLevel**() with weight λ_{sl} **Step 3** becomes:

```
$\textbf{max\_delta\_score} = \textbf{MinPossScore}$
for each $n_d \in \textbf{LandingNodes}(P_{\text{local}})$
    for each $t_i \in R(n_d)$
        $n_s = \textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_node}$
        $t_f = \textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{takeoff\_time}$
        $\textbf{path\_delta\_score} =$
            $\textbf{PathCache}(P_{\text{local}}, t_i, n_d).\textbf{path\_score} -$
            $(\textbf{Trace}(t_i, n_d).\textbf{score} - \textbf{Trace}(t_f, n_s).\textbf{score})$
        $\textbf{path\_sent\_delta\_score} =$
            $\lambda_{sl}(\textbf{SentLevel}(P_{\text{new}}) - \textbf{SentLevel}(P_{\text{cur}}))$
        if $(\textbf{path\_delta\_score} + \textbf{path\_sent\_delta\_score} > \textbf{max\_delta\_score})$
            $\textbf{max\_delta\_score} = \textbf{path\_delta\_score} + \textbf{path\_sent\_delta\_score}$
```

One important example of a sentence-level knowledge source is a long-distance language model. As an example, the BU sentence-level mixture language model [2] has been incorporated into the lattice local search. This model uses a mixture distribution of n different trigram models which are combined at the sentence level.

That is, the trigram probability of an entire hypothesis using each model is determined and the log of the weighted combination of these probabilities is the score for the entire hypothesis. In order to implement this model into the lattice local search, backward trigram models were built for each distribution (since the lattices are backwards). Then the arcs of each lattice were annotated with the n scores corresponding to the log trigram probability of each mixture LM ($n = 5$ in this case).

At each iteration of the local search, the total sentence-level mixture LM score was evaluated and weighted with its optimized weight from N-best rescoring. For each local path, another weighted sentence mixture LM score was calculated for the corresponding sentence hypothesis. The difference between the weighted sentence-level mixture LM scores for the whole sentence with local path and the initial hypothesis was then added to the delta score for the path obtained from the other knowledge sources and this score was used to determine the best local path in the local neighborhood.

Chapter 6

Experimental Results

This chapter presents experimental results for the three different lattice-based search algorithms and the older BU N-best recognizer. The speed and performance of the lattice N-best, lattice DP and lattice local search algorithms are given for the ARPA 1993 WSJ and Rutgers 1994 Switchboard recognition tasks. In addition, some important conclusions and observations are made about these results. Results are also presented using the BU sentence-level mixture model with the lattice N-best and lattice local search algorithms.

6.1 Paradigm

Recall that the BU N-best recognition system described in Chapter 3 that was used prior to this thesis work required phonetic segmentations of each N-best hypothesis to constrain the search space for the SSM. This older system, which will be referred to as the "old BU N-best recognition system" throughout this chapter, required a fifth pass of the BBN decoder to rescore and resegment each N-best hypothesis using the BBN HMM. The lattice N-best system, on the other hand, uses the phonetic segmentations on the N-best lattices for each N-best hypothesis. Since the N-best lattices are created after the fourth pass of the BBN decoder, the fifth pass of the BBN decoder is not needed. Thus the lattice N-best system is inherently more computationally efficient than the old N-best system that relies on the fifth HMM decoding pass.

It is important to verify that the lattice N-best system performs at least as well and as efficiently as the old BU N-best system. Therefore, comparable experiments for the lattice N-best and older N-best systems are reported for the WSJ H2-P0 and H1-P0 tasks. N-best segmentations are not available for the Rutgers Switchboard task, so results are not reported on this task for the older BU N-best system.

Table 6.1: *Size of N for the N-best lists and lattices generated for each test.*

WSJH2P0	WSJH1P0	Switchboard
100	500	2,000

The N-best lattices used for each experiment were generated so that the lattices correspond directly to an N-best list as described in Chapter 3. The size of N for each test is different. Table 6.1 gives N for each test reported on in this chapter. For WSJ-H2-P0, N = 100 was chosen to be consistent with the size of N used in previous experiments with the older BU N-best system. For H1-P0, N = 500 was chosen because the vocabulary and word error rates are greater on this test condition than they are for H2-P0. For Switchboard, N = 2,000 was picked because the word error rates are much higher (around 50%) for this task in general. However, weights were estimated for the Switchboard experiments using only the top 500 hypotheses because rescoring and estimating weights for N of 2,000 was considered too expensive. For both H2-P0 and H1-P0, weights were estimated using all hypotheses in the N-best lists.

Table 6.2: *Top 1-best word error rates for the three different tasks.*

Test	WSJ-H2-P0	WSJ-H1-P0	Switchboard
DEV	9.2%	18.9%	56.8%
EVAL	8.3%	16.7%	53.8%

The word error rates for the top 1-best answer in the N-best lists are presented in Table 6.2. Note that H2-P0 has the lowest 1-best error rates while Switchboard has the highest. In each case, the 1-best error rates provide a sense of how difficult each task is and provide a baseline for the the performance of each system discussed in this chapter.

The reason the three lattice-based search algorithms are evaluated on three different tasks is to determine the robustness of the algorithms for different size lattices and for different performance regions. The size of N that was chosen for each task is based on the performance region of the task. For instance, since H2-P0 has the lowest 1-best error rate of all the tasks, a relatively small N was used for its N-best lattices. On the other hand, since Switchboard has the highest error rate of all the tasks, a relatively large N was used. Consequently, it is expected that the speed and error rates for each algorithm on the three different tasks will scale accordingly: the algorithms are expected to run the fastest and have the lowest error rates on H2-P0, and the algorithms are expected run the slowest and have the highest error rates on Switchboard.

The lattice inclusion rates and coverage statistics are presented in Table 6.3. The sentence inclusion rate is the percentage of the time that the correct hypothesis is in the lattices, while the word coverage rate is 100% minus the minimum attainable word

Table 6.3: *Lattice inclusion and coverage rates for the development test sets of each task.*

	H2-P0 (N=100)	H1-P0 (N=500)	SWBD (N=2000)
Sentence Inclusion	79%	53%	10%
Word Coverage	97.8%	93.3%	64%

error rate for the lattices. Note that H2-P0 has the highest inclusion rate and the highest coverage while Switchboard has the lowest inclusion rate and lowest coverage.

The knowledge sources used for the WSJ H2-P0 development and evaluation tests include the SSM, the BBN trigram LM and the number of words, phones and inter-word silences. The SSM was trained on all of the WSJ acoustic training data, and the BBN trigram LM was trained on the WSJ0 LM training data and additional LM data processed by BBN. The knowledge sources used for H1-P0 are the same as those used for H2-P0 except that the BU trigram LM is used [2]. This LM was trained on the WSJ 1994 LM training data. For H1-P0, results are also reported using the BU sentence-level mixture LM which was also trained on the WSJ 1994 LM training data. The approximate HMM scores in that N-best lattices were not used in these experiments.

The knowledge sources used for the Switchboard experiments include the SSM, the BBN HMM, the BBN trigram LM and the number of words, phones, and inter-word silences. The SSM was trained on a subset of the BBN Switchboard training set based on the segmentations generated by J. Odell[1] [28]. The HMM scores used are approximate scores as described in Section 3.3.

[1]An older version of the segmentations than the ones released on the CD-ROM were used for these experiments

Weights for the lattice N-best, lattice DP and lattice local search algorithms are identical and were estimated using the optimization procedure described in Section 3.2. The lattice N-best algorithm was used to generate the scores for the optimization.

The speeds reported in this chapter for each algorithm are based on the number of CPU seconds required for decoding a set of utterances divided by the number of seconds of speech in the utterances. Therefore, each figure is the "number of times real time" that it takes to decode speech for some algorithm. The machines used for obtaining these figures were SUN Sparc 20 50's, Sparc 20 51's and Sparc 20 61's. The Sparc 20 50 has a 50 MHz clock speed, the Sparc 20 51 also has a 50 MHz clock speed and 1 Megabyte of cache RAM and the Sparc 20 61 has a 60 MHz clock speed and 1 Megabyte of cache RAM. The speed rates are all adjusted to be approximately equal to speed of the algorithms on a Sparc 20 50.

6.2 N-best Rescoring

Recall that the lattice N-best search rescores the top N sentence hypotheses by constraining the lattice DP algorithm to paths through the lattice corresponding to one N-best hypothesis at a time. The lattice N-best algorithm provides an important baseline for the lattice DP and lattice local search algorithms. The lattice N-best algorithm also provides a means of estimating weights for the other two lattice based algorithms. The lattice N-best algorithm differs from the old N-best algorithm in that the segmentation times (and therefore the DP time constraints) are different, and the HMM scores (when used) are produced by a different model.

In this section, the lattice N-best results are compared to the old N-best algorithm and the relative performance of each are compared. Results are only presented for the WSJ tasks since the appropriate data is not available to run the old system

Table 6.4: *H2-P0 results for the old N-best and lattice N-best algorithms (N=100).*

	Old N-best		Lattice N-best	
	Speed	Error Rate	Speed	Error Rate
DEV	11.3	7.5%	13.5	7.4%
EVAL	13.9	6.2%	13.9	6.2%

Table 6.5: *H1-P0 results for the old N-best and lattice N-best algorithms (N=100).*

	Old N-best		Lattice N-best	
	Speed	Error Rate	Speed	Error Rate
DEV	20.3	16.0%	12.1	15.7%
EVAL	23.6	14.6%	13.2	14.3%

on the Switchboard task. The speed and performance for H2-P0 are presented in Table 6.4 while the speed and performance for H1-P0 are presented in Table 6.5. All experiments are based on the search algorithm which does not use word score caching because little gain was observed in word caching experiments.

Note that the lattice N-best system performs slightly better than the old N-best system on H1-P0. This is probably due to the differences in the way the time constraints are calculated as discussed in Chapter 3. In general, the time constraints for an unmarked node in the triphone-expanded lattice will depend on all paths passing through that node. Since the lattice represents a large set of hypotheses, there will generally be more paths passing through an unmarked node in the triphone-expanded lattice used in the lattice N-best algorithm than there will be for an unmarked node in the triphone-expanded network used in the old N-best algorithm. This is because the triphone-expanded network represents only one N-best hypothesis while the triphone-expanded lattices represents all hypotheses. Therefore, the time constraints for unmarked nodes in the triphone-expanded lattices of the lattice N-best algorithm may tend to be more liberal than the time constraints for unmarked nodes in the triphone-expanded networks of the old N-best algorithm.

The speeds of the lattice N-best and old N-best algorithms for H2-P0 presented in Tables 6.4 are comparable. However, the old N-best algorithm has the additional computational cost of requiring each N-best hypothesis to be individually resegmented by the HMM. This makes the lattice N-best algorithm inherently more efficient. For H1-P0, on the other hand, the old N-best algorithm is a factor of 1.8 times slower than the lattice N-best algorithm. Since the size of the N-best list is the same for both H1-P0 and H2-P0 in this case, the lattice N-best speed seem reasonable, and the increase in cost for the old N-best algorithm is an anomalous result that we cannot yet explain.

Note that the old N-best algorithm is a factor of 1.8 times slower on H1-P0 than on H2-P0. This could be because H1-P0 is a larger vocabulary task (20,000 words vs. 5,000 words) and the N-best lists may represent a higher degree of variation. If this is the case, then the N-best lists for H1-P0 would have a corresponding larger number of unique triphones than the N-best lists for H2-P0. Consequently, triphone segment and distribution caching would not save as much computation.

Table 6.6: *Lattice N-best results and speeds for H1-P0 test for N=500.*

	WSJ-H1-P0	
	Speed	Error Rate
DEV	87.4	15.6%
EVAL	94.6	14.3%

The N-best results presented in this section indicate that the lattice N-best algorithm is comparable (or better) in performance and speed to the old N-best algorithm. This establishes that the lattice N-best algorithm is a valid baseline for the lattice DP and lattice local search algorithms. For reference, lattice N-best results for H1-P0 with N equal to 500 are presented in Table 6.6. Note that the speed is a factor of 10 times slower in this case than it is for N equal to 100, and there is no performance gain. N-best results for Switchboard will be presented later in this chapter.

6.3 Local Search Variations

In this section, the speed and performance of the local search algorithm are examined under different conditions. First, two different definitions of the local neighborhood are investigated: a local neighborhood with three types of local paths and a local neighborhood with six types of local paths. Then the effects of word score caching on speed and performance are examined.

Table 6.7: *WSJ H2-P0 development and evaluation results for the lattice N-best and lattice local search algorithms using three and six types of local paths without word caching.*

	Lattice N-best		Lattice Local Search 3 local paths		Lattice Local Search 6 local paths	
Test	Speed	Error Rate	Speed	Error Rate	Speed	Error Rate
DEV	13.5	7.4%	3.8	7.8%	4.6	7.3%
EVAL	13.9	6.2%	3.8	7.4%	3.7	6.2%

Table 6.7 presents the results of the lattice N-best and lattice local search algo-

rithm on the WSJ H2-P0 tests for the case when the local neighborhood includes only the "deletion," "substitution," and "split" local paths and the case where the local neighborhood includes the "insertion," "deletion," "substitution," "split," "merge," and "double substitution" local paths. Note that the lattice local search algorithm is about 1.2 times slower when six types of local paths are used. However, in this case the results are as much as 14% better than the corresponding results with the smaller local neighborhood. Therefore, the local neighborhood that includes all six local paths seems to be a better choice.

Table 6.8: *WSJ H1-P0 development and evaluation results for the lattice N-best and lattice local search algorithms using six types of local paths with word caching.*

	Lattice N-best		Lattice Local Search	
Test	Speed	Error Rate	Speed	Error Rate
DEV	83.4	15.6%	8.7	15.9%
EVAL	89.2	14.3%	9.8	14.5%

The results for using word caching with the local neighborhood that has six types of local paths for H1-P0 are given in Table 6.8. In this case, the speed is increased by only a small amount (less than 5%) for the lattice N-best algorithm. The speed is the same for the lattice local search algorithm. Note that the word caching does not change the performance of either algorithm.

The remaining results reported in this chapter will use the lattice local search algorithm which has six types of local paths in its local neighborhood. In addition, word caching will not be used in either the lattice N-best or lattice local search experiments.

6.4 Search Tradeoffs with Markov Assumptions

Table 6.9: *WSJ H2-P0 development and evaluation results for the three different search algorithms.*

	Lattice N-best		Lattice Local Search		Lattice DP	
Test	Speed	Error Rate	Speed	Error Rate	Speed	Error Rate
DEV	13.5	7.4%	4.6	7.5%	4.6	7.3%
EVAL	13.9	6.2%	3.7	6.5%	4.1	6.2%

Table 6.10: *WSJ H1-P0 development and evaluation results for the three different search algorithms.*

	Lattice N-best		Lattice Local Search		Lattice DP	
Test	Speed	Error Rate	Speed	Error Rate	Speed	Error Rate
DEV	103	15.6%	8.6	15.9%	8.0	15.4%
EVAL	95	14.3%	9.8	14.5%	9.1	13.9%

Table 6.11: *Switchboard development and evaluation results for the three different search algorithms.*

	Lattice N-best		Lattice Local Search		Lattice DP	
Test	Speed	Error Rate	Speed	Error Rate	Speed	Error Rate
DEV	133	56.3%	7.5	57.1%	15.4	58.0%
EVAL	133	54.7%	7.5	54.6%	15.4	56.2%

This section presents results for the lattice N-best, lattice DP and lattice local search algorithms on the WSJ H2-P0, WSJ H1-P0 and Switchboard tasks for the case where only Markov knowledge sources are used. Both speed and performance are reported for each algorithm on each task in Tables 6.9, 6.10 and 6.11.

For both WSJ tasks, it was found that the lattice DP has performance slightly better than the lattice N-best and is a factor of 3 to 11 times faster depending on the size of N. For H2-P0, N is equal to 100 and the lattice DP algorithm is about 3 times faster than the lattice N-best algorithm. For H1-P0, N is equal to 500 and the lattice DP algorithm is about 11 times faster.

This increase in speed for the lattice DP algorithm over the lattice N-best algorithm is also observed on the Switchboard task, but not the improvement in accuracy. There are two possible reasons for this. One is that the SSM acoustic model may not be as good as the HMM for the Switchboard task, in which case considering more hypotheses in the lattice would actually hurt performance. The second possible reason is that the weights, which were estimated on the top 500 hypotheses, are not optimized well enough for the full lattice which corresponds to an N of 2,000.

For all three tasks, the lattice local search algorithm gives performance only slightly worse than the lattice N-best algorithm with increases in speed of 3 to 18 times that of the lattice N-best algorithm depending on the size of the lattice. The difference in speed the lattice local search and the lattice DP algorithms depends on the size of the lattices. For small lattices where the lattice local search algorithm scores most of the lattice, the lattice local search algorithm is slower than the lattice DP algorithm. For H1-P0, comparing the N=100 lattice N-best to the N=500 lattice DP, we find that we can save more than a factor of 2 in computation and reduce the error rate by 4-5% through using the lattice DP algorithm. For the large Switchboard lattices, the lattice local search is actually faster than the lattice DP algorithm, but this result is flawed in that the rescoring SSM knowledge source is not as powerful as the orignal HMM.

The lattice local search algorithm converges in about 1.8 steps on the 5,000 word H2 test and about 2.5 steps on 20,000 word H1 test. The faster convergence on H2 is consistent with the lower error rates on this task and the smaller size of the lattices. Recall that for H2 N=100 was used whereas N=500 was used for H1.

6.5 Search Tradeoffs for a Sentence–Level Language Model

Table 6.12: *WSJ H1-P0 development and evaluation results for lattice local search algorithm with the BU mixture LM.*

	Lattice N-best		Lattice Local Search	
Test	Speed	Error Rate	Speed	Error Rate
DEV	103	15.4%	9.0	15.7%
EVAL	95	13.7%	9.6	13.5%

Results are presented for rescoring with the BU sentence-level mixture LM in Table 6.12. Note that the performance of the lattice local search algorithm is within 2% of the performance of the lattice N-best algorithm. In fact, the lattice local search algorithm performs slightly better on the evaluation test set than the lattice N-best algorithm.

The speed of the lattice local search algorithm using the sentence-level mixture LM is about the same as the speed of the algorithm without this model. The reason for this is that the sentence-level mixture LM requires relatively little computation compared to the SSM. Comparing the performance of the lattice local search with the sentence-level mixture model to the lattice DP without the sentence-level model (Table 6.10) it is seen that the lattice local search provides a means of improving recognition performance with sentence-level knowledge sources without significantly increasing computation.

6.6 Summary

The results presented in this chapter indicate that the lattice DP and lattice local search algorithms exhibit comparable performance to N-best rescoring when the same weights are used for combining knowledge sources. However, the results also demonstrate that the lattice DP and lattice local search algorithms are significantly faster than than the N-best algorithm by factors of 3 to 18 depending on the size of the lattice. The results also show that the lattice N-best algorithm performs at least as well as the old N-best algorithm, and thus provides a compact representation that is useful in weight estimation as well as search.

These results also show that, in general, a larger N leads to an increase in the relative computational efficiency of the lattice DP and lattice local search algorithms compared to the lattice N-best algorithm. Also, the lattice local search becomes more efficient than the lattice DP algorithm when the lattices are larger. The lattice local search has the additional advantage of allowing sentence-level knowledge sources to be incorporated into the search.

Therefore, if large lattices and sentence-level knowledge sources are going to be used for recognition, then the lattice local search algorithm is a good choice. However, if small lattices are going to be used with only Markov knowledge sources, then the lattice DP algorithm is a good choice. In both of these cases, the lattice N-best algorithm is critical in training mode in order to generate weights to use for the DP or local search.

The choice of the local neighborhood which has six types of local paths appears to be a good one. The lattice local search algorithm is both efficient and accurate using this neighborhood definition. A larger local neighborhood could be used but this would increase the computational complexity of the algorithm. On the other hand, a smaller local neighborhood leads to a decrease in performance.

Chapter 7

Conclusion

In this thesis, three different lattice-based search algorithms have been investigated: the lattice N-best algorithm, the lattice DP algorithm and the lattice local search algorithm. The lattice DP and lattice local search algorithms have been shown to be efficient alternatives to N-best rescoring. In addition, the lattice local search algorithm has been shown to be an effective means of incorporating sentence-level knowledge sources, such as the BU sentence-level mixture LM, into the search.

7.1 Summary

In order to rescore lattices with the BU SSM and sentence-level language model, the BBN decoder was modified so that N-best lattices could be produced. This involved saving information during the N-best traceback so that a lattice was created corresponding to the N-best list for each utterance. Since many other sites in the CSR community are also creating and potentially sharing lattices, a standard lattice file format specification was proposed for use by the community which is included in Appendix A along with some examples.

The lattice DP algorithm was presented as an efficient means of finding the hypothesis with the highest combined score in an N-best lattice using an optimized set of weights for combining the scores. In addition, it was shown that N-best rescoring could be easily accomplished using a special case of the DP algorithm where all paths through the lattice were turned off except for the paths corresponding to an N-best hypothesis. This algorithm was called the lattice N-best rescoring algorithm and it could be used as a means of estimating weights and as a performance baseline for the other lattice-based search algorithms.

It was shown that the lattice N-best rescoring algorithm performed at least as well as the older BU N-best rescoring algorithm while running just slightly faster. It was also shown that the lattice DP algorithm is much more efficient than the lattice N-best algorithm for Markov knowledge sources because it avoids redundant computation. The lattice DP algorithm was shown to perform as well, and in some cases better, than the lattice N-best algorithm while running up to 11 times faster for the particular cases evaluated here.

Then a novel lattice search approach, the lattice local search algorithm, was introduced. Although this algorithm was not guaranteed to find the globally optimal solution, it held the promise of being efficient and also of allowing for the incorporation of sentence-level knowledge sources into the search. Algorithms for caching and retrieving local path scores were presented, and finally a procedure was outlined for caching and retrieving word scores in either the lattice N-best or lattice local search algorithms. The lattice local search algorithm also proved to perform about as well as the lattice N-best algorithm while running as much as 18 times faster. The key advantage of the lattice local search algorithm over the lattice DP algorithm was shown to be its ability to incorporate the BU sentence-level mixture model into the search. Experiments showed that better results can be obtained by using the suboptimal local search with sentence-level knowledge sources than by using the optimal DP search with only Markov knowledge sources.

7.2 Future Work

The work presented in this thesis could be extended in many directions. The speed of the lattice DP and local search algorithms could be further improved. New ways of estimating score combination weights could be explored since the lattice N-best algorithm is very expensive. Also, new types of hybrid lattice search algorithms could be formulated. These possibilities are expanded on in this section.

Different approaches could be taken to speed up the different search algorithms. The lattice DP algorithm search time could be reduced by using a pruning strategy similar to that used in the standard beam search. The lattice local search algorithm could be speeded up by improving word score caching and by eliminating some of the resegmenting.

The local search algorithm could also be modified so that larger local neighborhoods are considered without significantly increasing the search time by using the HMM local path scores to determine which local paths to include in the local neighborhood. This could be done by including all paths that form one loop with the current solution but also have an HMM local path delta score that is within some fixed threshold. The threshold would have to be adjusted to control the size and inclusion rate of the corresponding local neighborhood.

The N-best lists corresponding to an N-best lattice are generally a sub-set of the paths through the lattices. Therefore, the weights obtained by optimizing scores from these N-best lists are not necessarily the best weights for combining the Markov knowledge sources used in the search. A better set of weights could be found by first scoring and annotating the lattices with the Markov knowledge sources using some initial set of weights and the lattice DP algorithm. Then candidate sets of weights would be generated by setting up a uniformly spaced grid in the weight space in a similar fashion to the approach used currently in the BU grid-based optimizer. The lattice error rate at each grid point would be evaluated by using DP on the lattices to find the highest scoring hypothesis for the given weights. Once the grid point which gives the lowest error rate is found, a more refined grid would be generated around this point and the algorithm would iterate until the error rate stays the same.

This approach would have the potential advantage of not using the lattice N-best algorithm which is much less efficient than the lattice DP algorithm although the weight optimization step would be more expensive. The trade-off between the cost increase in the optimization step and the cost decrease in the scoring step has to be assessed in experiments. We conjecture that performing DP with a candidate set of weights will not be expensive since scores are not actually calculated at this stage: scores are simply looked up from the lattice and multiplied by the weights. Moreover, the overall word error rate could potentially be reduced since weights could be optimized on thousands of hypotheses for an utterance.

A similar approach could be used for optimizing weights on the lattices if sentence-level knowledge sources are being used. This would require performing a local search on the lattices with a candidate set of weights for combining the scores on the lattice with the scores from the sentence-level knowledge sources. Then the weights which give the lowest over-all error rate would be used for performing recognition with the lattice local search. In fact, the weights obtained by optimizing with the lattice local search may be better suited to performing recognition with this algorithm.

The lattice local search algorithm could be used in cases where rescoring is going to be performed with some kind of sentence-level model after the lattices have been rescored with an acoustic model by using a two-stage algorithm. First the lattices would be annotated with the best acoustic score for each word after the lattice DP algorithm has been performed with the Markov knowledge sources. Then, during the local search, each path in the local neighborhood would be evaluated by combining the score from the sentence-level knowledge source with the Markov scores from the lattice. This would provide a fast means of developing and evaluating sentence-level knowledge sources such as long distance language models.

The two stage lattice–based algorithm has the additional advantage of starting the local search with a better initial solution. That is, instead of using the top N-best hypothesis from the BBN decoder as the initial starting point, the optimal path given all of the Markov knowledge sources and associated score combination weights is used. In theory, a better initial solution can increase the accuracy and speed of the lattice local search algorithm. The efficiency of the two stage algorithm would depend on the computational complexity of the knowledge sources used used in each stage. For instance, the two stage lattice algorithm would be a good choice for the BU SSM and sentence-level mixture LM used in this work because the computation for the sentence-level LM is small compared to the SSM.

Clearly there is more exploring to be done in the area of lattice search. As more powerful and expensive acoustic and language model are applied to harder recognition tasks, the search problem will become more important. The lattice search algorithms presented in this thesis provide a means of handling these situations both in present and future applications.

Bibliography

[1] R. Lau, R. Rosenfeld and S. Roukos, "Trigger-Based Language Models: a Maximum Entropy Approach," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* Vol. II, pp. 45-48, April 1993.

[2] R. Iyer, M. Ostendorf and J. Rohlicek, "Language Modeling with Sentence-Level Mixtures," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

[3] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, Jan. 1986, pp. 4-16.

[4] M. Ostendorf and S. Roukos, "A Stochastic Segment Model for Phoneme-Based Continuous Speech Recognition," *IEEE Trans. on Acoust., Speech, and Signal Proc.,* Dec. 1989, pp. 1857-1869.

[5] S. Roucos, M. Ostendorf, H. Gish and A. Derr, "Stochastic Segment Modeling Using the Estimate-Maximize Algorithm," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* pages 127–130, New York, New York, April 1988, pp. 127-130.

[6] D. Bertsekas, "Dynamic Programming, Deterministic and Stochastic Models," Prentice-Hall, 1987.

[7] F. Alleva, X. Huang and M. Hwang, "An Improved Search Algorithm Using Incremental Knowledge for Continuous Speech Recognition," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* April 1993, pp. 307-310.

[8] D. Paul, "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* March 1992, pp. 25-28.

[9] J. Odell, V. Valtchev, P. Woodland and S. Young, "A One Pass Decoder Design for Large Vocabulary Recognition," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

[10] S. Austin, R. Schwartz and P. Placeway, "The Forward-Backward Search Algorithm," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* March 1991, pp. 697-700.

[11] R. Schwartz and Y. Chow, "The N-Best Algorithm: An Efficient and Exact Procedure for Finding The N Most Likely Sentence Hypotheses," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* Apr. 1990, pp. 81-84.

[12] R. Schwartz and Y. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* Apr. 1990, pp. 701-704.

[13] M. Ostendorf, A. Kannan, S. Austin, O. Kimball, R. Schwartz and J. R. Rohlicek, "Integration of Diverse Recognition Methodologies Through Reevaluation of N-Best Sentence Hypotheses," *Proc. DARPA Workshop on Speech and Natural Language,* Feb. 1991, pp. 83-87.

[14] M. Ostendorf, F. Richardson, S. Tibrewal, R. Iyer, O. Kimbal, J. R. Rohlicek, "Stochastic Segment Modeling for CSR: The BU WSJ Benchmark System", *ARPA Spoken Language Systems Technology Workshop,* March 1994

[15] M. Bates *et al.*, "The BBN/HARC Spoken Language Understanding System," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* April 1993, Vol. II pp. 111 - 122.

[16] M. Rayner, D. Carter, V. Digalakis and P. Price, "Combining Knowledge Sources to Reorder N-Best Speech Hypothesis Lists," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

[17] R. Grishman, *et al.*System description for Nov. 1994 ARPA WSJ benchmark.

[18] A. Kannan, M. Ostendorf and J. Rohlicek, "Weight Estimation for N-Best Rescoring," *Proc. DARPA Workshop on Speech and Natural Language,* Feb. 1992, pp. 455-456.

[19] M. Ostendorf, A. Kannan, O. Kimball and J. R. Rohlicek, "Continuous Word Recognition Based on the Stochastic Segment Model", *Proc. DARPA Workshop on Continuous Speech Recognition,* Sept. 1992.

[20] L. Nguyen, R. Schwartz, Y. Zhao and G. Zavaliagkos, "Is N-Best Dead?," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

[21] H. Murveit, J. Butzberger, V. Digalakis and M. Weintraub, "Large-Vocabulary Dictation Using SRI's DECIPHER^{TM} Speech Recognition System: Progressive Search Technique," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* Apr. 1993, pp. 319-322.

[22] J. Odell and P. Woodland, proposed lattice file format specification for the ARPA CSR community.

[23] F. Alleva, X. Huang and M. Hwang, "An Improved Search Algorithm Using Incremental Knowledge for Continuous Speech Recognition," *Proceedings Int'l. Conf. on Acoust., Speech and Signal Proc.,* Apr. 1993, pp. 307-310.

[24] C. Papadimitriou and K. Steiglitz, "On The Complexity of Local Search for the Traveling Salesman Problem," *SIAM Journal of Computing,* Mar. 1977, pp. 76-83.

[25] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp.671-680, 1983.

[26] D. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, 1989.

[27] V. Digalakis, M. Ostendorf and J. R. Rohlicek "Fast Algorithms for Phone Classification and Recognition Using Segment-Based Models," *IEEE Trans. on Signal Proc.,* Dec. 1992, pp. 2885-2896.

[28] Rutgers "Frontiers in Speech Recognition" 1994 workshop CD-ROM.

[29] F. Kubala *et al.*, "The Hub and Spoke Paradigm for CSR Evaluation," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

[30] D. Pallett *et al.*, "1993 Benchmark Tests for The ARPA Spoken Language Program," *Proc. ARPA Workshop on Human Language Technology,* March 1994.

Appendix A

Lattice File Format

This section describes the lattice file format specification proposed for use by the ARPA CSR community. This file specification is designed to allow for defining anything from a finite state grammar to a word lattice with time, phone segmentation and language and acoustic model score information. Each node of the lattice either represents a word or is a "NULL node" while the arcs represent transitions between words.

File structure:

The lattice files consist of 3 sections: a header section, a node specification section and an arc specification section. Each section is terminated by a single '>' character occurring at the beginning of a line. A line beginning with a '*' character is a comment line which is intended as a means of providing useful information for humans that are reading the file.

1.0 Header Section:

Each line of the header section contains a label followed by one or more values. The labels consist of any alpha-numeric characters and '_'. Upper vs. lower case is significant. The values can be strings, ints, floats or a combination of the three. There are some special labels which must appear in the header section as well as other predefined optional ones. Both will be discussed momentarily. Other user-defined labels can be added as needed but will be ignored by default.

1.1 MANDATORY LABELS

The mandatory labels are:

Labels	Value type
FF_VERS	float
UTTERANCE	string
N_NODES	int
N_ARCS	int
FIRST_NODE	int
LAST_NODE	int
DIRECTION	string
NODE_SPEC	list of strings
ARC_SPEC	list of strings
WORD_LOC	string

FF_VERS:

The value of this label corresponds to the file format version. The first version of this file format will be 1.0 proceeded by 1.1, etc. Major revision will increment the part of the number before the decimal point.

UTTERANCE:

The value of this label is a string corresponding to the identifier of the utterance that the lattice was generated for. This should be the standard utterance identifier as specified for the corpus that is being used (i.e. full path name to data files shouldn't be used).

N_NODES:

The value of this label is the number of nodes in the lattice.

N_ARCS:

The value of this label is the number of arcs in the lattice.

FIRST_NODE:

The value of this label is the index of the first node of the lattice.

LAST_NODE:

The value of this label is the index of the last node of the lattice.

DIRECTION:

The value of this label indicates the direction of the lattice. The meaningful values this label can have are:

Value	Meaning
forward	lattice is in the forward direction
backward	lattice is in the backward direction

NODE_SPEC:

The value of this label is a list of string identifiers which indicate the format for the node specification section. Essentially, each node is specified by a single line where each column of the line contains a different value for the node. The NODE_SPEC label indicates what values for each node are present and what order these values occur in. Some values should only be defined if the value of the WORD_LOC label is NODES:

Value	Type	Meaning
INDEX	int	The corresponding column contains the unique index of the node between 0 and N_NODES-1.
WORD	string	The transcription of the word associated with this node. This field should not be defined if WORD_LOC is equal to ARCS. The string "#" means the node is a NULL node. In general, the lattice should begin and end with a NULL node.

Other values NODE_SPEC can have (if this information is provided as indicated in the value of NODE_SPEC):

Value	Type	Meaning
TIME	float	The end time for the segmentation of the word The value of the optional label TIME must also be defined (see OPTIONAL LABELS below).
SEG	string	This is a sequence of times and phone labels which correspond to the phonetic segmentation of the word node. The format is a sequence of <start time>:<phone label>: A single ":" is used to indicate that the word has no segmentation (i.e. if it's a NULL node). The value of the label TIME must also be defined (see OTHER LABELS below). This field should not be defined if WORD_LOC is equal to ARCS.
AC_SCORE	float	Acoustic score of this word ending at the above time. The value of the label AC_LOG_BASE must also be defined (see OTHER LABELS below). This field should not be defined if WORD_LOC is equal to ARCS.

ARC_SPEC:

The value of this label takes on a list of string identifiers which indicate the format for the arc specification section. Essentially, each arc is specified by a single line where each column of the line contains a different value for the arc. The ARC_SPEC label indicates what values for each arc are present and what order these values occur in.

Value	Type	Meaning
INDEX	int	The corresponding column contains the unique index of the arc from 0 to N_ARCS-1.
S_NODE	int	The index of the source node for this arc.
T_NODE	int	The index of the target node for this arc.
WORD	string	The transcription of the word associated with this arc. This field should not be defined if WORD_LOC is equal to nodes. The string "#" means the arc is a NULL arc.
PRON	int	Pronunciation version number. WORD_LOC must be NODES.

Other optional values ARC_SPEC can have:

Value	Type	Meaning
LM_SCORE	float	The LM score corresponding to the transition from the S_NODE to the T_NODE. The value of the label LM_LOG_BASE must also be defined (see OTHER LABELS below).
SEG	string	This is a sequence of times and phone labels which correspond to the phonetic segmentation of the word arc. The format is a sequence of <start time>:<phone label>: A single ":" is used to indicate that the word has no segmentation (i.e. if it's a NULL arc). The value of the label TIME must also be defined (see OTHER LABELS below). This field should not be defined if WORD_LOC is equal to NODES.
AC_SCORE	float	Acoustic score of this word ending at the time of the target node. The value of the label AC_LOG_BASE must also be defined (see OTHER LABELS below). This field should not be defined if WORD_LOC is equal to NODES.
PRON	int	Pronunciation version number. WORD_LOC must be ARCS.

WORD_LOC:

Location of the word names (and segmentations if they are provided). Possible values are:

Value	Meaning
NODES	Words (and segmentations) appear on nodes
ARCS	Words (and segmentations) appear on arcs

<u>1.2 OTHER LABELS:</u>

Labels	Value type
AC_LOG_BASE	string
LM_LOG_BASE	string
TIME	float
AC_WT	float
LM_WT	float
WRD_WT	float
PHN_WT	float
SIL_WT	float

AC_LOG_BASE:

If acoustic scores are provided, then this label must be included. The value of the label is the logarithm base for acoustic scores. This is actually a string with meaningful values:

Value	Meaning
e	natural log
10	log base 10
<number>	log base <number>
-	No log is taken

LM_LOG_BASE:

If LM scores are provided, then this label must be included. The value of the label is the logarithm base for language model scores. The values this label can have are the same as for the label AC_LOG_BASE.

TIME:

If any time information is included in the lattice, then this label must be defined. The value of this label is the number of seconds corresponding to one unit of time. For example:

Value	Meaning
1	Time is in seconds
0.01	Time is in 10ms frames

AC_WT, LM_WT, WRD_WT, PHN_WT, SIL_WT:

The weights for combining the acoustic model, LM, <# of words>, <# of phones> and <# of silences> scores. These weights were used during decoding to produce the lattice.

2.0 Node Specification Section

Each line of the node specification section defines the information for each node of the lattice. The meaning of each column in each line is defined by the value of the label NODE_SPEC defined above. The minimum node specification corresponds to a unique index for the node (from 0 to N_NODES-1) and an orthographic transcription of the word (where "#" means the node is a NULL node). Optionally, time information, phone segmentations and acoustic scores can be included.

3.0 Arc Specification Section

Each line of the arc specification section defines the information for each arc of the lattice. The meaning of each column in each line is defined by the value of the label ARC_SPEC defined above. The minimum arc specification corresponds to a unique index for the arc (from 0 to N_ARCS-1), the index of the arc's source node and the index of the arcs target node. Optionally, LM scores can be provided. An example of the N-best lattices used in this work is given below in the proposed lattice file format specification. Note that '*' indicates a comment line.

```
*
* This is a comment line
* FF_VERS = File Format Version
FF_VERS 1.0
*
* Identifier string for the utterance:
*
UTTERANCE 4kac020j
N_NODES 16
N_ARCS 19
FIRST_NODE 0
LAST_NODE 15
WORD_LOC NODES
```

```
*
* Log base "-" means logs aren't used, "e" means natural logs
*
AC_LOG_BASE 10
LM_LOG_BASE 10
DIRECTION backward
*
* Time in secs/unit, i.e. 10ms frames are 0.01.
*
TIME 0.01
*
* Now specify what each column in the node specification section represents
*
NODE_SPEC INDEX TIME WORD SEG AC_SCORE
*
* And specify what each column in the arc specification section represents
*
ARC_SPEC INDEX S_NODE T_NODE LM_SCORE
*
* Weights for score combination
AC_WT 1.0
LM_WT 2.4
WRD_WT 0.0
PHN_WT 0.0
SIL_WT 0.0
*
* Header ends with a single '>' at the beginning of the line:
```

```
*
>
*
* Next, nodes are defined:
* word label "#" means the node is a null node.
* Node specification is: <node ID> <time> <word label> <segmentation> <acoustic
score>
*
0 2.89 # : 0
1 2.89 OUT 254:T:257:-: -88.3425
2 2.24 TAPPED 196:AE:210:P:219:T:224:AW: -190.708
3 1.82 BASICALLY 133:EY:147:S:154:IX:157:K:168:L:174:IY:182:T: -226.499
4 1.26 ARE 126:B: -22.1332
5 1.19 CONSUMERS 60:AX:64:N:69:S:82:Y:91:UW:94:M:99:AXR:110:Z:119:AXR: -
252.609
6 1.26 AS 124:Z:129:B: -33.649
7 1.19 CONSUMERS 60:AX:64:N:69:S:82:Y:91:UW:94:M:99:AXR:110:Z:119:AE: -245.505
8 1.26 TO 121:AX:125:B: -39.9698
9 1.19 CONSUMERS 60:AX:64:N:69:S:82:Y:91:UW:94:M:99:AXR:110:Z:118:T: -238.046
10 1.26 AND 123:N:126:D:129:B: -35.2405
11 1.19 CONSUMERS 60:AX:64:N:69:S:82:Y:91:UW:94:M:99:AXR:110:Z:119:AX: -
240.626
12 1.26 HAVE 120:AE:123:V:128:B: -42.8524
13 1.19 CONSUMERS 60:AX:64:N:69:S:82:Y:91:UW:94:M:99:AXR:110:Z:117:HH: -
234.304
14 0.60 </sil> 0:-:50:K: -168.064
15 0.00 # : 0
```

```
*
* Arc specifications end with a single '>' at the beginning of the line:
*
>
*
* Arc specification is: <arc ID> <start node ID> <end node ID> <LM score>
*
0 14 15 0
1 13 14 -0.781652
2 12 13 -3.03728
3 3 12 -2.05463
4 2 3 -1.65264
5 1 2 -4.71225
6 11 14 -1.74197
7 10 11 -3.64027
8 3 10 -2.59062
9 9 14 -2.34496
10 8 9 -3.6626
11 3 8 -1.94296
12 7 14 -1.38464
13 6 7 -3.70726
14 3 6 -1.7643
15 5 14 -0.870984
16 4 5 -3.3276
17 3 4 -1.09431
18 0 1 -2.79161
```

Here is an HTK lattice converted to this format:

```
* Lattice generated by CU-HTK 23 Feb 94
*
* File : “/data/wsj/wsj1/si_dt_20/4k0/4k0c030t.wv2”
*
* Best hypothesis “!SENT_START IT DIDN’T ELABORATE !SENT_END” Score=-
20218.25
*
* Language model scores from “/lib/baseline-lm/bg-boc-lm20o.nvp”.
* Dictionary used “/lib/dictionaries/dragon/wsj.sls”.
* Acoustic scores from “/models/htk2/hmm11”.
*
* Header
*
FF_VERS 1.0
UTTERANCE 4k0c030t
N_NODES 24
N_ARCS 39
FIRST_NODE 0
* This is the only node without any out arcs so it must be the last one:
LAST_NODE 23
WORD_LOC ARCS
AC_LOG_BASE e
LM_LOG_BASE e
AC_WT 1.0
```

```
LM_WT 16.0
WRD_WT 0.0
DIRECTION forward
TIME 1.0
NODE_SPEC INDEX TIME
ARC_SPEC INDEX S_NODE T_NODE WORD PRON AC_SCORE LM_SCORE
>
0 0.00
1 0.25
2 0.26
3 0.61
4 0.62
5 0.62
6 0.71
7 0.72
8 0.72
9 0.72
10 0.72
11 0.72
12 0.72
13 0.73
14 0.78
15 0.78
16 0.80
17 0.80
18 0.81
19 0.81
```

```
20 1.33
21 2.09
22 2.09
23 2.85
>
0 0 1 !SENT_START 0 -1432.27 0.00
1 0 2 !SENT_START 0 -1500.93 0.00
2 0 3 !SENT_START 0 -3759.32 0.00
3 0 4 !SENT_START 0 -3829.60 0.00
4 1 5 TO 3 -2434.05 -87.29
5 2 5 TO 1 -2431.55 -87.29
6 4 6 AND 3 -798.30 -69.71
7 4 7 IT 0 -791.79 -62.05
8 4 8 AND 2 -836.88 -69.71
9 3 9 BUT 0 -965.47 -51.14
10 4 10 A. 0 -783.36 -105.95
11 4 11 IN 0 -835.98 -49.01
12 4 12 A 0 -783.36 -59.66
13 4 13 AT 0 -923.59 -77.95
14 4 14 THE 0 -1326.40 -27.96
15 4 15 E. 0 -1321.67 -121.96
16 4 16 A 2 -1451.38 -59.66
17 4 17 THE 2 -1490.78 -27.96
18 4 18 IT 0 -1450.07 -62.05
19 5 18 IT 0 -1450.07 -110.42
20 6 18 IT 0 -775.76 -85.12
21 7 18 IT 0 -687.68 -125.32
```

```
22 8 18 IT 0 -687.68 -85.12
23 9 18 IT 0 -687.68 -50.28
24 10 18 IT 0 -689.67 -108.91
25 11 18 IT 0 -706.89 -113.78
26 12 18 IT 0 -689.67 -194.91
27 13 18 IT 0 -619.20 -100.24
28 4 19 IT 1 -1567.49 -62.05
29 14 20 DIDN'T 0 -4452.87 -195.48
30 15 20 DIDN'T 0 -4452.87 -118.62
31 16 20 DIDN'T 0 -4303.97 -189.88
32 17 20 DIDN'T 0 -4303.97 -195.48
33 18 20 DIDN'T 0 -4222.70 -78.74
34 19 20 DIDN'T 0 -4235.65 -78.74
35 20 21 ELABORATE 2 -5847.54 -62.72
36 20 22 ELABORATE 0 -5859.59 -62.72
37 21 23 !SENT_END 0 -4651.00 -13.83
38 22 23 !SENT_END 0 -4651.00 -13.83
```